UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR


Doble Grado en Ingeniería Informática y Matemáticas

# TRABAJO DE FIN DE GRADO

## Motor de juego como plataforma para la aplicación de algoritmos de Inteligencia Artificial

Óscar Manuel Losada Suárez

Tutor: David Camacho Fernández

Julio 2015

## Agradecimientos

Quiero aprovecharme de la costumbre de encabezar textos de este tipo con agradecimientos para expresar las debidas gracias que generalmente no he dado.

A mi tutor David, por aceptar dirigirme este trabajo cuando acudí a él tardíamente y por estar siempre atento a su bandeja de entrada, incluso en las épocas más inverosímiles, y responderme siempre con atención.

A Juan, por regalarme tan a menudo la media hora que me llevaba a mi explicarle el problema que me atascaba y los cinco minutos que le llevaba a él pensar y explicarme solución.

A Fede, con quien disfruté sobremesas extendidas de las que salí siempre con más y mejores ideas que las que traía.

A los miembros de «El Pozo», mis compañeros en estos años mal llevados.

A mi familia, por entender tan bien como es posible el silencio y saber combinar el látigo con el mimo en la dolida recta final.

## Abstract

In recent times an interest in the investigation of Artificial Inteligence through videogames has emerged. Areas such as procedural content generation, the creation o artificial agents capable of playing games competently or the creation of agents whose conduct is undistinguishable from that of a human player attract a growing amount of researchers nowadays. Research in the área of Artificial Intelligence in games not only can improve the quality of videogames in the future, but also exploits the possibilities of the medium, which provides an environment for the simulation of problems that are analogous to those of the real world.

Until now, research has been done mainly using comercial videogames that weren't developed with the purpose of being used this way. This project aims to develop a platform that has different characteristics from those offered by the games that have been used in research until now and that provides a convenient interface for the users so that they can implement Artificial Intelligence agents that can be tested within the game.

The platform has been developed using the *Unity3D* game engine and is formed by a map generator, a tactical strategy team-based multiagent *3D* game, an *API* that allows the agents to interact with the game and a few demonstration agents to show the use of the platform. The game has been designed with the intention of allowing the exploration of aspects of gameplay such as terrain analysis, cooperation between independent, heterogeneous agents, the formation of hierarchies within the teams and the communication of information between agents.


**Keywords:** Videogames, intelligent agents, software platform, *Unity3D*, Articial Intelligence.

## Resumen

En el pasado reciente ha surgido un interés por la investigación de la Inteligencia Artificial a través de los videojuegos. Temas como la generación procedural de contenido, la creación de agentes que puedan jugar a un videojuego de forma competente o de agentes cuya conducta sea indistinguible de la de un jugador humano atraen a una cantidad creciente de investigadores en la actualidad. La investigación en el ámbito de la Inteligencia Artificial aplicada a videojuegos no sólo puede mejorar la calidad de los videojuegos creados, si no que aprovecha las posibilidades del medio, que provee un entorno de simulación de problemas análogos a los del mundo real.

La investigación hasta ahora se ha realizado mayoritariamente utilizando videojuegos comerciales no desarrollados con el propósito de ser usados de esta forma. Este proyecto pretende desarrollar una plataforma que ofrezca características diferentes de las que ofrecen los juegos que han sido usados en la investigación hasta ahora y que ofrezca una interfaz cómoda para que los usuarios puedan implementar agentes de Inteligencia Artificial que puedan ser probados en el juego.

La plataforma se ha desarrollado sobre el motor de juego *Unity3D* y está compuesta por un generador de mapas, un juego de estrategia táctica multiagente por equipos en *3D*, una *API* que permita a los agentes desarrollados por los usuarios interactuar con el juego y algunos agentes de prueba para demostrar el uso de la plataforma. El diseño del juego se ha realizado con la intención de que permita la exploración de aspectos de juego como el análisis del terreno, la cooperación entre agentes independientes y heterogéneos, la formación de jerarquías y la comunicación de información entre los agentes.

**Palabras clave:** Videojuegos, agentes inteligentes, plataforma software, *Unity3D*, Inteligencia Artificial.

## Glosario y terminología

Agentes / jugadores / Inteligencias Artificiales (IAs) Se usan estos términos para referirse a los algoritmos que toman decisiones inteligentes dentro del juego haciendo las veces de jugador.

Avatar Se refiere a los personajes virtuales que controlan los *agentes*.

Mesh / malla En gráficos *3D* de ordenadores se refiere a una colección de vértices, aristas y caras que definen una forma de un objeto polihédrico.

# Índice

## Índice de cuadros



## Índice de figuras

# Parte I. Introducción

En los albores de la investigación en el campo de la Inteligencia Artificial (IA), el énfasis estaba en crear máquinas que «pensasen» como los humanos. Sin embargo, la idea del *Test de Turing* [1] hizo que se pasara a una aproximación de «caja negra», más centrada en los resultados. Así, a día de hoy existe un cierto consenso en el campo y lo que se pretende es crear agentes artificiales que tomen decisiones racionales o «inteligentes» cuando se les presenta con un problema.

La investigación de la Inteligencia Artificial en los videojuegos comenzó entorno al juego *Pac-Man*, especialmente en su versión *Ms. Pac-Man*, después de que investigadores como J. Koza y J. Rosca lo propusieran como un entorno interesante que planteaba un problema de priorización de tareas y permitía la aplicación de algorítmos genéticos [2]. El interés de la comunidad investigadora por la investigación en los videojuegos ha ido en aumento desde entonces, llegándose a celebrar competiciones para desarrollar Inteligencias Artificiales que jueguen a *Ms. Pac-Man* [3, 4] y a *Infinite Mario Bros.* [5] en años recientes.

Los juegos en general, y los videojuegos en particular, plantean problemas análogos a otros encontrados en la vida humana. Dejando de lado su valor como forma de entretenimiento, los juegos son poderosas herramientas para el aprendizaje, pues eliminan la presencia de riesgos y consecuencias reales y facilitan la exploración libre, por parte de los jugadores, de los problemas que modelan. A principios del siglo XIX, el Barón von Reiswitz, y después su hijo, miembros del ejército prusiano, diseñaron el juego *Kriegsspiel*, que llegó a ser usado para entrenar a los oficiales del ejército prusiano [6, 7].


Fig. 1: Versión de Kriegsspiel de la Fundación Berlín-Brandenburg

Concretamente los videojuegos ofrecen un entorno al investigador que requiere unos recursos mucho menores de los que serían necesarios normalmente para desarrollar ideas de Inteligencia Artificial. Asimismo, no son otra cosa que simuladores, que se pueden construir de manera que

modelen los problemas que interese abordar. Además de esto, no hay que menospreciar el hecho de que los videojuegos son una industria importante hoy en día [26]. Una parte muy importante de ellos en muchos casos es la Inteligencia Artificial que controla a parte de los agentes participantes, de modo que estudiar cómo hacer mejores Inteligencias Artificiales para los juegos repercutirá en el aumento de su calidad.

## 1. Objetivos

El objetivo de este trabajo es crear una plataforma para el desarrollo y prueba de algoritmos de Inteligencia Artificial. Es decir, lo que se pretende es diseñar un problema, que constituirá el juego, lo suficientemente interesante para que sirva de entorno de pruebas para jugadores artificiales creados por los usuarios y proporcionar herramientas para que los usuarios puedan inyectar sus algoritmos en el juego.

El problema será un juego de acción táctica en tiempo real en 3D con dos equipos integrados por varios jugadores, cada uno de los cuales será controlado por un agente de Inteligencia Artificial. Más concretamente, el juego está basado en el deporte *Paintball*, en el que los jugadores deben tratar de marcar a sus oponentes con bolas de pintura disparadas con pistolas de aire comprimido, sin embargo, detalles estéticos como la representación de la pintura o los sonidos se dejan para desarrollo posterior, pues este trabajo se centra en la creación de una plataforma funcional.

La intención de este proyecto es que pueda ser utilizado por investigadores y desarrolladores de Inteligencia Artificial de forma semejante a como se han usado otros juegos en eventos como el IEEE CIG (Computational Intelligence and Games) [4, 8] o en CEC (Conference on Evolutionary Computation) [3]. Sin embargo, estos proyectos partían ya de productos comerciales con una larga evolución y pruebas extensivas para mejorarlos, por lo que este trabajo se concibe, no sólo como algo de una escala menor, si no como algo que podría seguir desarrollándose y mejorándose basándose en las observaciones que se hagan con su uso.

En conclusión, los objetivos del trabajo son los siguientes:

- Diseñar las reglas de un juego e implementarlo.
- Desarrollar un generador de mapas para usar en el juego.
- Crear una API para la incorporación de agentes de Inteligencia Artificial creados por usuarios a la plataforma.
- Desarrollar algunos agentes de Inteligencia Artificial de prueba que sirvan para demostrar el funcionamiento de la plataforma.
- Llevar a cabo pruebas con los agentes desarrollados y analizar los resultados.

## 2. Estado del arte

En el ámbito de la Inteligencia Artificial aplicada a videojuegos, tradicionalmente se han explorado tres vías:

- **Jugadores inteligentes**, que consiste en crear agentes que reciben información del estado del juego, del entorno, a través de sensores, evalúan dicha información y actúan en función de ella a través de actuadores que pueden afectar al estado del juego.
- **Creación procedural de contenido** (*Procedural Content Generation*, PCG), consistente en crear un algoritmo que genere contenido, generalmente niveles o mapas, que se considere «interesante».
- ***Test de Turing***, en el que se trata de crear un agente que juegue a un juego de forma que sea indistinguible de un humano jugando.

Dentro de la categoría de creación de jugadores inteligentes se han usado los juegos *Ms. Pac-Man* en *Ms. Pac-Man AI Competition* y *Ms. Pac-Man vs Ghosts Competition* [3, 4], *Infinite Mario Bros.* en *Mario AI Competition* [5], y su sucesor *SuperTux* en *Platformer AI Competition* [9] y *StarCraft* en *StarCraft AI Competition* [8]. Es en esta categoría en la que se centra este proyecto, apodado *PaintBol*, e igual que en los anteriores se usaron técnicas como *máquinas de estados* y *behaviour trees*, *redes neuronales*, *algoritmos evolutivos* y de *aprendizaje automático*, *lógica difusa*, *influence maps*, *reinforcement learning* y métodos de *Monte Carlo*, en este proyecto también se espera que diversas técnicas de Inteligencia Artificial sean aplicables.

En las otras dos categorías se han usado *Infinite Mario Bros.* en *Mario AI Competition* [5] y *SuperTux* en *Platformer AI Competition* [9] en creación procedural de contenido y *Unreal Tournament 2004* en *2k BotPrize* [10] e *Infinite Mario Bros.* en *Mario AI Competition* [5] en en las pruebas del *Test de Turing*. Estas dos categorías también serían aplicables a *PaintBol*, pero no se han implementado herrramientas específicas para facilitar su aplicación.

Otro proyecto de interés en el ámbito es *CodinGame* [11], que ofrece una plataforma con varios mini-juegos en la que es posible probar implementaciones de agentes directamente online e incluso competir contra las implementaciones de otros usuarios en tiempo real. Uno de los miembros del equipo desarrollador de CodinGame [12] propone las siguientes características para clasificar los juegos a la hora de definir el tipo de problema que plantean a los agentes:

- **Conocido** o **desconocido**: dependiendo de si se sabe a priori como va a reaccionar el entorno a las acciones del agente. Por ejemplo, un recién nacido evoluciona en un entorno desconocido, que tiene que descubrir, y por eso prueba absolutamente todo.
- **Accesible** o **inaccesible**: dependiendo de si los agentes tienen acceso a toda la información del entorno que podría usarse para tomar decisiones. Por ejemplo, en el Póker, los jugadores no tienen sólo información parcial del estado del juego, pues no conocen las cartas de los demás jugadores.
- **Determinista** o **no determinista**: dependiendo de si el comportamiento del entorno esta definido lógicamente o probabilísticamente. Por ejemplo, en el Ajedrez, se sabe con certeza la consecuencia que tendrá una determinada acción, por lo que es un juego determinista, mientras que en el Póker, las cartas que le tocan a cada jugador son aleatorias, por lo que es un juego no determinista.

- **Estático** o **dinámico**: dependiendo de si el entorno cambia mientras que los agentes estan tomando decisiones. Por ejemplo, para un agente que controle un coche, alguien podría aparecer en su camino mientras el agente esté aún pensando qué hacer.
- **Discreto** o **continuo**: dependiendo de si hay un número finito o infinito de acciones posibles para el agente.
- **Solitario** o **multijugador**: dependiendo de si intervienen uno o más agentes en el juego.

Utilizando estos criterios, podemos clasificar los juegos anteriormente mencionados y este proyecto:

| *Ms. Pac-Man* | *Infinite Mario Bros.* | *SuperTux* | *StarCraft* | *Unreal Tournament 2004* | *PaintBol* |
|---|---|---|---|---|---|
| Conocido | Conocido | Conocido | Conocido | Conocido | Conocido |
| Accesible | Accesible | Accesible | Inaccesible | Inaccesible | Inaccesible |
| No determinista | Determinista | Determinista | Determinista | Determinista | No determinista |
| Dinámico | Dinámico | Dinámico | Dinámico | Dinámico | Estático / Dinámico |
| Discreto | Discreto | Discreto | Continuo | Continuo | Continuo |
| Multijugador | Solitario | Solitario | Multijugador | Multijugador | Multijugador |

Tab. 1: Clasificación de juegos usados para investigación de IA usando los criterios de Loick Michard

*PaintBol* es un juego de tipo **conocido** porque las reglas están preestablecidas y son conocidas por los agentes, por ejemplo, si alcanzas a un adversario con un ataque, sabes que quedará deshabilitado durante unos segundos y después reaparecerá en otro punto aleatorio del mapa. Es **inaccesible**, porque los agentes tienen sólo información parcial del estado de la partida en cada momento. Es **no determinista** porque al comenzar la partida y cuando son alcanzados por un ataque, los jugadores aparecen en posiciones aleatorias del mapa y porque al disparar existe un factor aleatorio que influye en que el disparo alcance a su objetivo o no. El juego es **estático** o **dinámico** en función de como sean los algoritmos de los agentes: si todas las decisiones se toman de forma atómica en menos de un frame, entonces sería estático, pero los agentes pueden tener memoria y usar varios «turnos» para tomar una única decisión, en ese caso sería dinámico. Como el espacio en el que actuan los agentes es continuo hasta donde permite la capacidad de representación de los ordenadores, el juego es de tipo **continuo**. Finalmente, es **multijugador** por definición, pues el problema consiste precisamente en que varios agentes se enfrenten en el contexto de las reglas del juego.

Con todo, esta clasificación no permite señalar algunas características interesantes que distinguen a *PaintBol* de los otros juegos:

- El entorno tridimensional en el que se desarrolla el juego, junto con la relevancia de la topografía de los mapas y la distribución de los objetos del terreno para las mecánicas del juego hacen que el **análisis del terreno** (*terrain reasoning*) sea una herramienta útil e interesante para los agentes [13]. Para aumentar las posibilidades de la plataforma, esta cuenta con una herramiento de generación procedural de mapas, que evita que se puedan diseñar estrategias basadas en las características específicas de mapas concretos y además implica que las características de los escenarios en los que se desarrolla la acción podrían redefinirse y ampliarse en el futuro. De los juegos anteriores, sólo en *StarCraft* y *Unreal Tournament 2004* es tan relevante este aspecto, pero en ambos los mapas son conocidos a priori.

- Los equipos están formados por agentes posiblemente **heterogéneos** e independientes. Son heterogéneos en el sentido de que los algoritmos que controlan a cada uno de los agentes pueden ser completamente diferentes, y son independientes porque todos los agentes tienen información parcial no compartida del estado del juego en función de su posición, la dirección en la que estén mirando, etc... Por ejemplo, los agentes en principio no saben donde están sus aliados - ni sus enemigos - ni qué están haciendo a no ser que estén dentro de su arco de visión. Estas características permiten que se formen estrategias basadas en **jerarquías**, que dan lugar a situaciones interesantes cuando, por ejemplo, parte de un equipo queda fuera de la partida y los agentes restantes podrían saber reaccionar a esa situación. De los otros juegos anteriores, de nuevo, esta es una característica sólo compartida por *Unreal Tournament 2004* y *StarCraft*, aunque en el segundo caso, no era una posibilidad explotada en *StarCraft AI Competition*, pues los enfrentamientos estaban restringidos a uno contra uno.
- Los agentes tienen la capacidad de **comunicar información** que posean del estado de la partida, por ejemplo avistamientos de adversarios o aliados, localizaciones de zonas ventajosas estrategicamente en el mapa, etc, que hayan descubierto o que otros agentes les hayan transmitido. En conjunción con la caracterísica anterior, este aspecto facilita y promueve la aparición de comportamientos emergentes al nivel de los equipos, y esta característica no es compartida por ninguno de los juegos anteriores, pues en *StarCraft*, aunque los bandos están integrados por muchas unidades, todas son controladas por un único agente que recibe información de todas sus unidades y las controla a todas.

## 3. Tecnologías a utilizar

Para el desarrollo del proyecto se han usado principalmente *Unity3D* [14] para el desarrollo de la aplicación, y en mucha menor medida *Blender* [15] para la creación de los avatares del los jugadores y de algunas de sus animaciones.

Los motivos para la elección de Unity3D como motor de juego son múltiples:

- Desde su versión 5.0, toda sus funcionalidad es de uso gratuito siempre que los beneficios anuales de la organización no superen una cierta cota máxima.
- Es multiplataforma, permite portar las aplicaciones, en general sin necesidad de cambio alguno a cualquier del sinfín plataformas que soporta: Web, *Windows*, *Linux*, *Mac*, *iOS*, *Android*, *BlackBerry*, *WindowsStore*, *Windows Phone 8*, *WebGL*, *Xbox 360*, *Xbox One*, *PS3*, *PS Vita* y *PS4*.
- Tiene una completa librería con soporte para *rendering*, sonido, físicas y controles.
- Tiene un editor que facilita la labor de depurado de las aplicaciones y que unifica el *content pipeline*, evitando la necesidad de utilizar otros subsistemas para gestión e importación de recursos y tiene un editor de niveles integrado.
- Existe una enorme comunidad de usuarios dispuesta a compartir su conocimiento, un gran número de tutoriales y una documentación completa de las librerías que ofrece.

- Permite la programación en *C#* - lenguaje ya conocido por el alumno - y sobre el framework *Mono*, que ofrece una amplísima cantidad de funciones que aceleran y facilitan el desarrollo.
- Se integra bien con *Blender*.

Por su parte, *Blender* es una herramienta utilizada con anterioridad por el alumno para la creación y animación de modelos 3D y de reconocida potencia, siendo además gratuito y de código abierto.

## 4. Estructura del documento

En la parte II se detalla el alcance del trabajo, se explica la metodología seguida y se especifican los requisitos de la plataforma.

La parte III contiene un resumen del diseño y la arquitectura de la plataforma y sus dos partes principales: el generador de mapas y el juego.

La implementación del proyecto se encuentra en la parte IV, explicando en mayor detalle algunos de los algoritmos y técnicas que conforman las partes más representativas del trabajo. En particular, en esta sección se explican también los agentes de *IA* implementados.

La parte V recoge las pruebas realizadas para la verificación y validación, así como los resultados de la pruebas experimentales realizadas con los agentes desarrollados.

Finalmente, la parte VI se centra en proponer posibles lineas de trabajo futuras que, después del trabajo realizado, se consideran importantes e interesantes.

# Parte II. Definición del proyecto

En esta sección se concretará el alcance, se explicará la metodología empleada y se detallará una relación de los requisitos funcionales y no funcionales del trabajo.

## 5. Alcance

Este trabajo tiene como objetivo ofrecer motor de que juego que conforme una plataforma para la investigación en el campo de la Inteligencia Artificial. Para esto, la plataforma se compondrá de una herramienta para la generación de mapas en *3D* y un videojuego en el que serán utilizados los mapas generados como escenario. Además, el videojuego contará con una *API* que dará acceso a los desarrolladores e investigadores usuarios de la plataforma a los sensores y actuadores que permitan a los agentes de Inteligencia Artificial que desarrollen interactuar con el juego.

Asimismo, se crearán tres jugadores inteligentes de prueba, con la intención de que sirvan de demostración de funcionamiento de la plataforma, pero que además se usarán para realizar un pequeño estudio de resultados y que servirán de punto de referencia y reflexión acerca de las debilidades y necesidades de mejora y ampliación de la plataforma.

Como se apunta en el párrafo anterior, el alcance de este trabajo no es el mismo que el del proyecto que se presenta. El desarrollo de una plataforma con las características que se desean supera con creces las posibilidades de un trabajo de fin de grado, y por tanto, no se pretende que el proyecto haya alcanzado en este punto la refinación de los videojuegos anteriormente usados en este área y mencionados en apartados anteriores. Siguiendo este razonamiento, se plantea este trabajo como el desarrollo de una primera versión, centrada en obtener un producto funcional, pero no perfecto. Se ha pospuesto la adición de algunos componentes estéticos en cuestión de gráficos y del sonido de la aplicación por no considerarse prioritarios y se dejan pendientes numerosas mejoras posibles que se han ido observando durante el desarrollo.

## 6. Metodología

Para el desarrollo de la aplicación se ha optado por seguir un ciclo de vida incremental iterativo. En un análisis inicial se detectó la posibilidad dividir el desarrollo en tres incrementos, a saber: el generador de mapas, el videojuego en sí y la creación y prueba de las IAs de demostración. Por otra parte, la limitada experiencia en el ámbito y con las tecnologías relevantes del alumno, motivó a emplear un planteamiento más próximo a las metodologías ágiles de desarrollo, con énfasis en la adaptabilidad a medida que se encontraban dificultades en el desarrollo, que a las tradicionales y más centradas en la predicción y la planificación.

En las siguientes figuras, se muestra el diagrama de Gantt del proceso seguido en el proyecto por incrementos, incluyendose las dependencias entre bloques y tareas. Después de cada figura se detallan las entradas, tareas y salidas de cada una de las fases de desarrollo en cada uno de los incrementos.

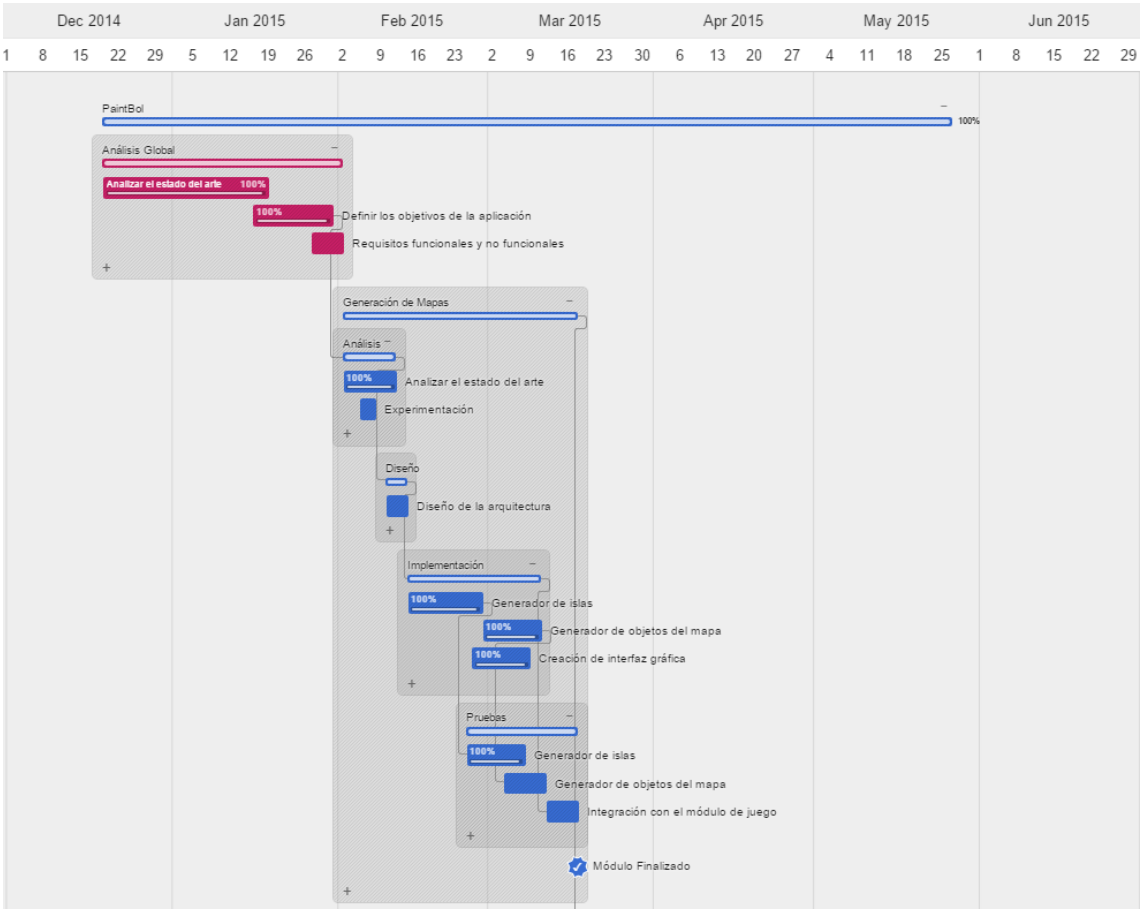



Fig. 2: Diagrama de Gantt del proyecto (Generación de mapas)

## Análisis Global

| Entradas | Tareas | Salidas |
|---|---|---|
| | Analizar el estado del arte<br><br>Definir objetivos de la plataforma<br>Especificar requisitos funcionales y no funcionales | Documento de visión o concepto del proyecto |

Tab. 2: Entradas, tareas y salidas del Análisis Global

## Generación de Mapas

| Fase | Entradas | Tareas | Salidas |
|---|---|---|---|
| Análisis | Documento de visión o concepto del proyecto | Analizar el estado del arte de la generación procedural de contenido<br>Explorar algunas de las técnicas utilizadas | Definición de objetivos de la herramienta de generación de mapas |
| Diseño | Objetivos de la herramineta de generación de mapas | Diseñar la arquitectura de la herramienta | Arquitectura de la herramienta |
| Implementación | Información sobre la arquitectura y el diseño de la herramienta | Codificar la herramienta de generación de mapas | Código de la herramienta<br><br>Documentación del código |
| Pruebas | Código de la aplicación | Realización de pruebas de verificación y validación<br><br>Preparación para la integración con el módulo de juego | Código de la herramienta verificado, validado y preparado para la integración inmediata con el juego |

Tab. 3: Entradas, tareas y salidas de la Generación de Mapas

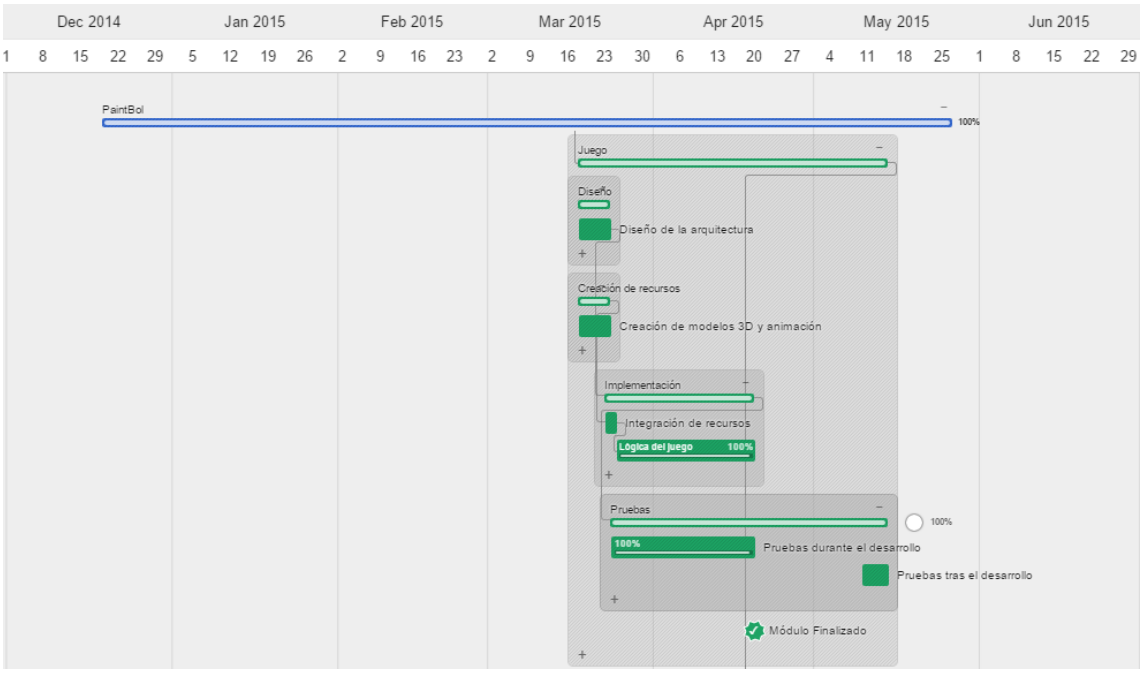



Fig. 3: Diagrama de Gantt del proyecto (Juego)

## Juego

| Fase | Entradas | Tareas | Salidas |
|---|---|---|---|
| Diseño | Documento de visión o concepto del proyecto | Diseñar la arquitectura del juego<br>Diseñar la interfaz gráfica de usuario<br>Diseñar la API para jugadores artificiales | Arquitectura del juego y de la API<br>Diseño de la interfaz gráfica de usuario |
| Creación de recursos | Documento de visión o concepto del proyecto | Creación y obtención de modelos 3D y animaciones de los avatares. | Modelos 3D y animaciones de los avatares |
| Implementación | Arquitectura del juego y de la API<br>Modelos 3D y animaciones de los avatares | Integración de los recursos<br><br>Codificación del juego y de la API | Código del juego y de la API<br><br>Documentación de la aplicación<br><br>Manual de usuario para desarrolladores de IA para la plataforma |
| Pruebas | Mapas generados por la herramienta de generación de mapas<br>Código del juego y de la API | Realización de pruebas de verificación y validación | Código del juego verificado y validado |

Tab. 4: Entradas, tareas y salidas del juego

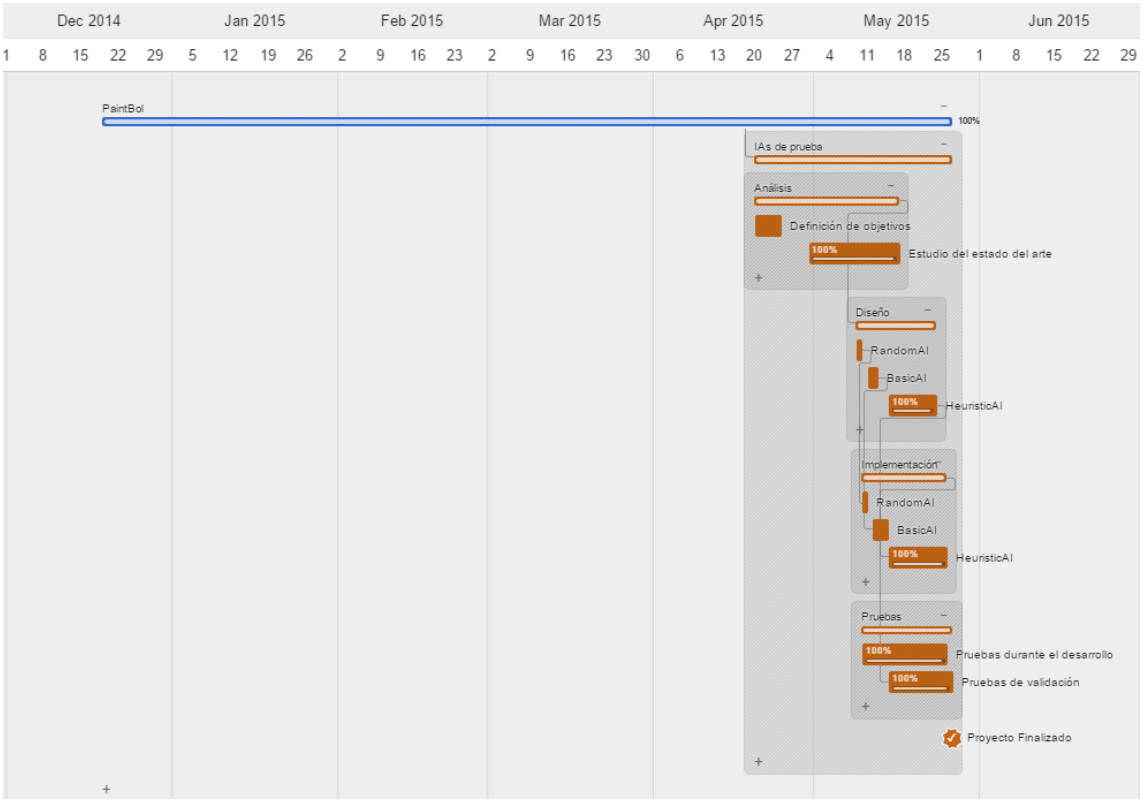



Fig. 4: Diagrama de Gantt del proyecto (IAs de prueba)

### Inteligencias Artificiales de prueba

| Fase | Entradas | Tareas | Salidas |
|---|---|---|---|
| Análisis | Documento de visión o concepto del proyecto<br>Código del juego y de la API | Definición de objetivos<br><br>Estudio del estado del arte | Objetivos del incremento |
| Diseño | Objetivos del incremento<br><br>API para jugadores aritificiales | Diseñar cada uno de los jugadores | Diseño de cada uno de los jugadores |
| Implementación | Diseño de cada uno de los jugadores | Codificación de cada uno de los jugadores | Código de cada uno de los jugadores<br>Actualización del manual de usuario |
| Pruebas | Código del juego<br><br>Código de los jugadores artificiales | Realización de pruebas de verificación y validación<br>Evaluación del sistema y planteamiento de mejoras futuras | Código verificado y validado<br><br>Resultados de los enfrentamientos entre los jugadores y de prueba desarrollados |

Tab. 5: Entradas, tareas y salidas de las Inteligencias Artificiales de prueba

## 7. Requisitos

A continuación se enumeran los requisitos que se definieron en la fase de análisis global sobre el problema planteado: crear una plataforma para la investigación de técnicas de Inteligencia Artificial a través de una videojuego. Este análisis se ha hecho mediante el estudio del estado del arte en el contexto de la investigación de IA en videojuegos. Se dividen los requisitos en funcionales y no funcionales.

### Requisitos funcionales

1. Se podrán generar proceduralmente mapas en *3D*. [16]
2. Los mapas tendrán obstáculos (*objetos del terreno*) que puedan obstruir la visión y servir como cobertura para los avatares.
3. Se deberá poder distinguir en el mapa que zonas son transitables y qué zonas no, así como distinguir la superficie del terreno de los *objetos del terreno*.
4. Se guardarán las dimensiones del mapa, así como la posición y dimensiones [21] de cada uno de los *objetos del terreno* que haya en los mapas que se generen.
5. Se ofrecerán parámetros al usuario que permitan ajustar las características de los mapas generados. En particular, se deberán poder ajustar la densidad de los objetos del terreno y las dimensiones del mapa.
6. Los jugadores podrán desplazarse por zonas transitables de la superficie del mapa.

7. El desplazamiento se podrá realizar de tres maneras: corriendo (máxima velocidad, máximo ruido), andando (velocidad intermedia, ruido intermedio) o agachados (velocidad mínima, ruido mínimo, aumenta la dificultad de impactarles con disparos y se reduce su visibilidad).

8. Los jugadores podrán estar en dos posturas (*stances*): de pie o agachados (aumenta la dificulta de impactarles con disparos y se reduce su visibilidad).

9. Habrá diez jugadores en cada partida.

10. Los jugadores estarán divididos en dos equipos rivales.

11. Los jugadores aparecerán en puntos aleatorios, pero transitables, del mapa.

12. Los jugadores podrán atacar a otros jugadores mediante tres mecanismos: un ataque de corto alcance (cuerpo a cuerpo), un ataque a distancia (disparos) y un ataque de área a distancia (granadas).

13. Los jugadores podrán disparar sólo a jugadores a los que estén apuntando.

14. Los jugadores podrán apuntar sólo a jugadores enemigos que estén dentro de su área de visión.

15. La probabilidad de impactar con un disparo dependerá de la postura del objetivo, del movimiento del objetivo, del movimiento del jugador que dispara, la parte del cuerpo a la que se decida apuntar y del tiempo invertido en apuntar.

16. La acción de disparar quedará deshabilitada durante un breve lapso de tiempo después de cada uso.

17. Los ataques cuerpo a cuerpo realizados por un jugador sólo afectarán a jugadores enemigos que se encuentren en el área de efecto del ataque.

18. Las granadas se lanzarán utilizando un ángulo de lanzamiento, una dirección de lanzamiento y una fuerza de lanzamiento.

19. Las granadas tendrán un retardo temporal desde el momento en que caigan al suelo después de ser lanzadas hasta el momento de su explosión.

20. Cuando las granadas exploten, todos los jugadores, independientemente de qué jugador las lance, que se encuentren en el área de efecto, serán afectados.

21. Los jugadores comenzarán las partidas con un cierto número de «puntos de vida». Si un jugador se queda sin puntos de vida dejará de participar en la partida.

22. Cuando un jugador sea alcanzado por un ataque, perderá un punto de vida y quedará deshabilitado durante un breve lapso de tiempo, durante el cual no podrá realizar ninguna acción ni ser alcanzado por ningún ataque y después del cual reaparecerá en un punto aleatorio del mapa.

23. Los jugadores recibirán información acerca de su entorno a través de dos sensores: la vista y el oído.

24. La visión de los jugadores está restringida a un arco frontal de un cierto radio. La orientación de dicho arco variará con el desplazamiento.

25. Cuando un jugador vea a otro jugador, sabrá que acción está realizando el jugador visto y conocerá su posición y la dirección en la que miraba en ese momento y sabrá si el jugador visto es un aliado o un adversario.

26. Cuando un jugador vea un *objeto del terreno* conocerá su posición y sus dimensiones.

27. Las acciones de los jugadores producirán ruido. El alcance del ruido producido será verosímil con la realidad.

28. El oído de los jugadores recibirá información de los ruidos que se produzcan lo suficientemente cerca de su posición.

29. El ruido informará a los jugadores que lo oigan de la dirección en la que se emitió y de la acción que lo produjo.

30. Los jugadores podrán comunicar la información que hayan recibido a través de sus sensores a otros jugadores (ruidos, objetos del terreno vistos y jugadores vistos).

31. El alcance del ruido producido por la acción de comunicarse podrá ser regulado por el jugador que comunica la información.

32. Cualquier jugador que oiga el ruido producido por una acción de comunicación recibirá la información que contenga dicha comunicación.

33. Los avatares serán controlados por agentes de inteligencia artificial (jugadores no humanos).

34. Se creará un sistema de puntuación para los equipos para evaluar las partidas.

35. La aplicación mantendrá actualizadas las puntuaciones de los equipos durante las partidas.

36. Se guardarán registros (*logs*) de las acciones de la partida para su análisis posterior.

37. El usuario deberá poder seleccionar un mapa de los creados por la herramienta de generación de mapas y un algoritmo de los disponibles para controlar a cada uno de los jugadores al comienzo de cada partida.

38. Durante el curso de la partida, el usuario deberá poder pausar la partida, ver la puntuación de cada equipo en ese momento, reanudar la partida y finalizar la partida.

39. La aplicación dará tiempo de ejecución periódicamente a cada uno de los jugadores para evaluar su entorno y tomar decisiones.

### Requisitos no funcionales

1. Los mapas deberán ser guardados de forma que sean incorporables al juego.
2. El juego tendrá una interfaz gráfica de usuario para operar la aplicación en conjunción con controles con el teclado y/o el ratón.
3. La aplicación incluirá una *API* que permita a las *IA*s implementadas por los usuarios interactuar con el juego.

# Parte III. Diseño

En este apartado se describe el diseño de la plataforma, explicándose la estructura general de las dos partes en las que se divide y justificándose las decisiones de diseño más relevantes.

## 8. Partes de la plataforma

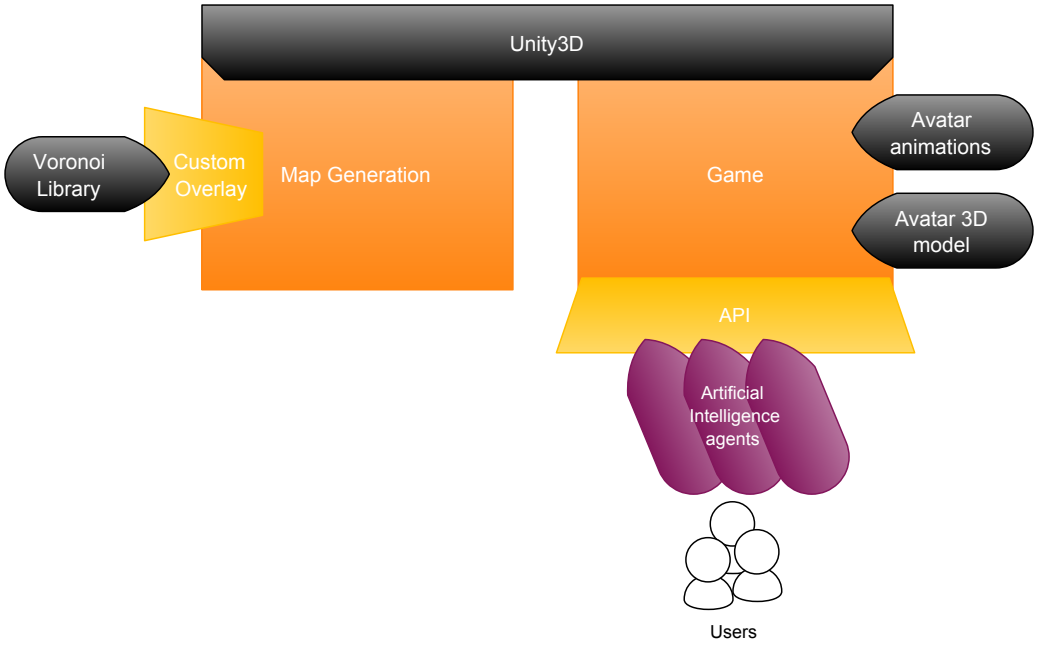



Fig. 5: Esquema de la arquitectura de la plataforma

Tras el análisis de los requisitos que se detallan en la sección anterior, se optó por dividir la plataforma en dos partes: el generador de mapas y el juego. La necesidad de tener un sistema de navegación para un entorno *3D* como es el de la plataforma y la complejidad de desarrollar uno propio hizo que se diese una elevada prioridad a poder usar el sistema de navegación proporcionado por *Unity3D* [20]. Este sistema permite crear de forma automática una malla de navegación (*NavMesh*) a partir del conjunto de *meshes* que definan el mapa y utilizándola, proporciona funciones que automatizan la navegación entre dos puntos cualesquiera del mapa. Sin embargo, el proceso de creación de las mallas de navegación sólo puede ser realizado desde el editor de *Unity3D* y no desde una aplicación ya compilada e independiente. Aunque en general esta limitación podría haber imposibilitado el uso del sistema, en este caso concluimos que las consecuencias negativas no serían tan grandes. Dado que la plataforma está pensada para que otros desarrolladores creen la Inteligencia Artificial que gobierne a los avatares en el juego y puesto que desarrollar un lenguaje de *scripting* en este punto era impensable por falta de recursos, en cualquier caso, los usuarios tendrían que compilar la plataforma incluyendo sus *IA*s usando *Unity3D*. Por tanto en esta misma fase se pueden generar los mapas con la herramienta de generación, crear sus mallas de navegación con el editor e

incorporar los mapas creados al juego.

## 9. Generación de mapas

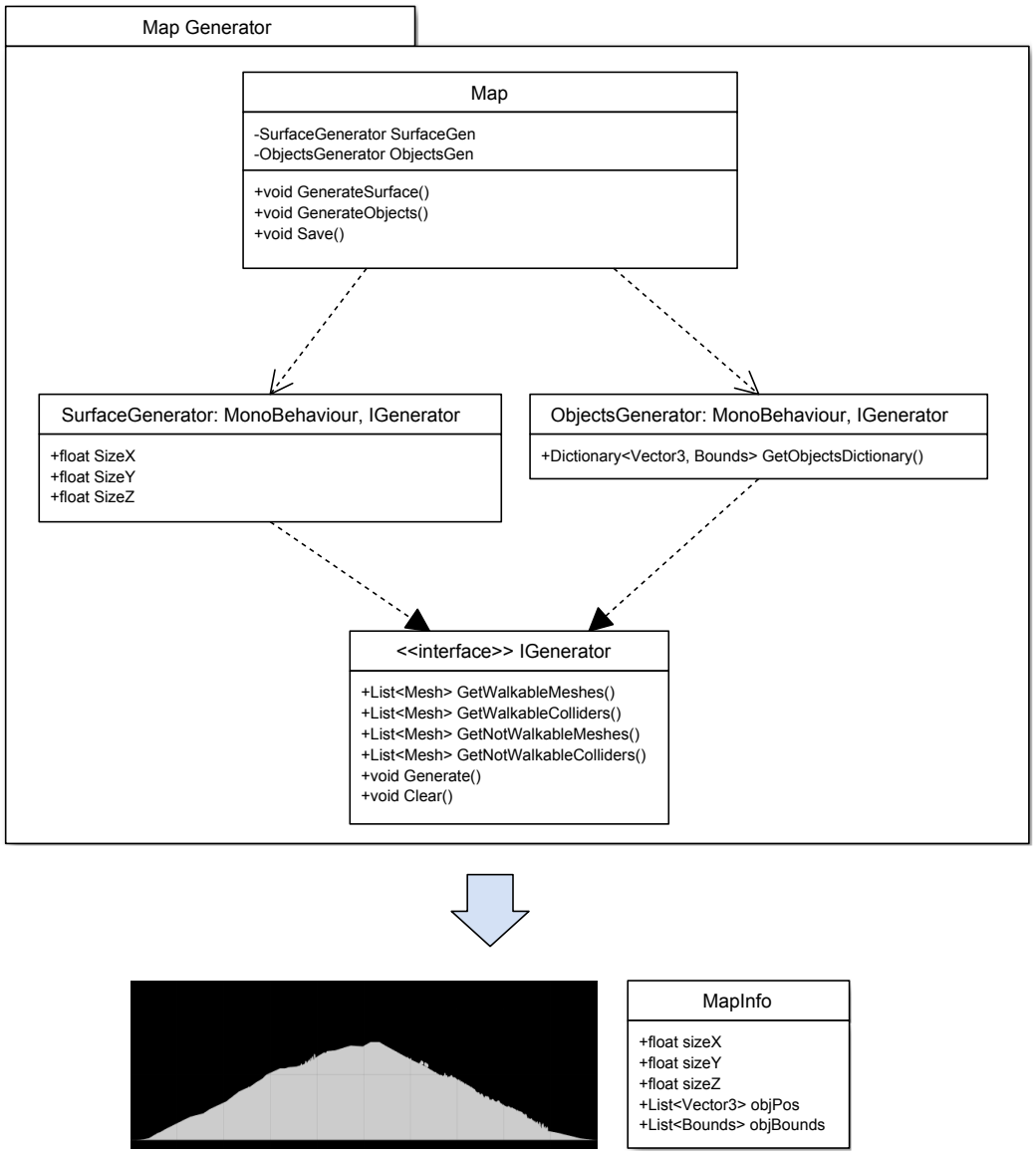



Fig. 6: Esquema de la arquitectura del generador de mapas

En los videojuegos, generalmente sólo es necesario mantener en memoria y mostrar en la pantalla la parte del mundo que está lo suficientemente próxima al jugador como para que este pueda

interactuar con ella. Un método comúnmente usado para ahorrar tiempo de procesamiento es sólo mantener activa y en memoria la parte del mundo relevante para el jugador en cada momento, es decir, aquella con la que pueda interactuar. En este caso, la presencia de diez agentes que estarán activos durante la partida exigiría para aplicar el método anterior un sistema muy elaborado, pues al mismo tiempo, era importante que el tamaño de los mapas fuese suficiente para dar lugar a mecánicas de exploración. Por estos motivos, se optó por una estética minimalista, basada en formas geométricas simples que permitiese una baja densidad de polígonos, y sin texturas, con la intención de minimizar el tamaño en memoria de los mapas y poder mantenerlos cargados en su totalidad durante las partidas.

La elección de esta estética con una geometría simple posibilitó además que los objetos del terreno fuesen lo suficientemente sencillos geométricamente como para construirlos proceduralmente también, lo que tiene la ventaja de eliminar la necesidad de adquirir y almacenar pesados recursos gráficos en el proyecto y además reduce el coste de tener objetos con variaciones en el tamaño y forma.

En la figura 6 se muestra un esquema de la arquitectura de la herramienta de generación de mapas. La interfaz `IGenerator`, junto con las clases abstractas `SurfaceGenerator` y `ObjectGenerator` definen los métodos y miembros que debe tener cualquier generador que se quiera usar con la clase `Map` para crear un mapa. En la herramienta se incluye un generador de superficies y otro de objetos del terreno, pero esta arquitectura general facilita la creación e incorporación rápida de otros generadores que sigan este esquema. La herramienta producirá entonces un mapa y una instancia de la clase `MapInfo`, que almacena la información exigida por el requisito funcional 4, que guardará en una *escena* de *Unity3D* [18].

## 10. Juego

El funcionamiento básico del juego se puede ver en la Figura 7. La clase `MatchHandler` se encarga de proporcionar a las instancias de la clase `Actor`, que representan a los personajes del juego la información que les proporcionan sus sentidos o *sensores* (vista y oído) y por otra parte se encarga de efectuar las acciones de los personajes sobre su entorno. Así, las partidas consisten esencialmente en un bucle en el que se alterna la recolección de información para proporcionar a las *IA*s que controlan a los personajes del juego con la cesión del control a dichas *IA*s para que analicen la información de que disponen y tomen decisiones.

Este último punto conecta con una decisión de diseño que se tomó para el desarrollo del juego y que merece la pena justificar. Actualmente, *Unity3D* no permite el uso de hilos (*threads*) a los desarrolladores que lo utilizan. Esta hubiese sido la técnica idónea para poder gestionar los turnos de los agentes de Inteligencia Artificial, pues permitiría controlar desde el hilo principal del juego el tiempo que se les permitía usar en cada ronda a los agentes para tomar decisiones y abortar los hilos de aquellos agentes que excediesen el tiempo permitido. Posiblemente, el motivo por el que *Unity3D* no ha dado prioridad en sus actualizaciones a soportar el uso de hilos es que ofrece en su lugar corutinas (*coroutines*) [22]. Las corutinas permiten que una función interrumpa su ejecución guardando su entorno de variables y la reanude más adelante. Desgraciadamente, en el caso de este

proyecto, esto es una limitación notable respecto a los hilos, pues no hay manera de interrumpir la ejecución de una función desde fuera de esta, tiene que se ella misma la que interrumpa la ejecución. La consecuencia de esto es que la plataforma debe delegar en los usuarios la responsabilidad de que los agentes que creen controlen su propio tiempo de ejecución e interrumpan la ejecución cuando su tiempo asignado haya concluido.

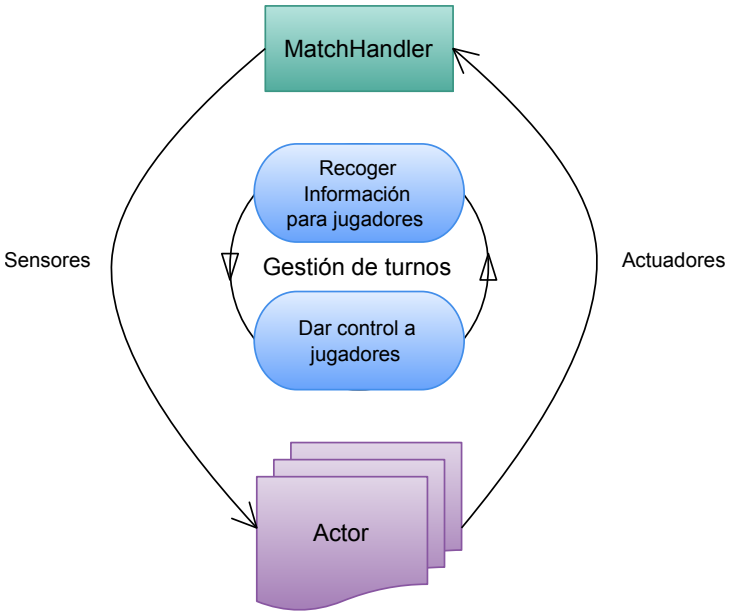



Fig. 7: Esquema de funcionamiento del juego

La Figura 8 describe a grandes rasgos la estructura del programa del juego. La clase MatchHandler implementará métodos ejecutables desde corutinas que permitan realizar el ciclo descrito de recolección de información del entorno de los personajes y distribución a los mismos y de cesión de control a los agentes. El uso de corutinas permitirá que la recolección se realice a lo largo de varios fotogramas (*frames*) [23] del juego.

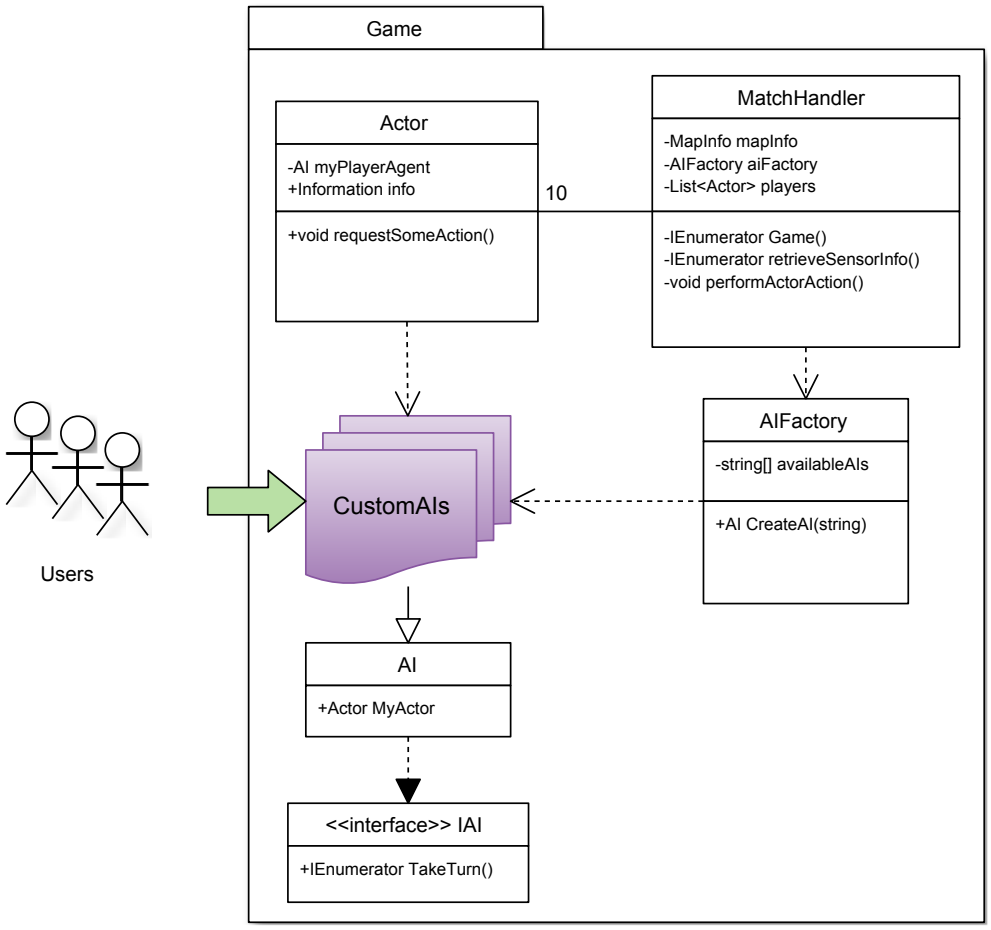



Fig. 8: Esquema de la arquitectura del juego

La clase `AIFactory`, en conjunción con la interfaz `IAI` y la clase abstracta `AI` definen los requisitos que deben tener los agente de IA implementados por los usuarios. El método `TakeTurn()` será el punto de entrada en cada turno para la ejecución de los agentes. Junto con estas, la clase `Actor` forma la *API* a la que tendrán acceso los usuarios para implementar sus agentes y será esta clase la que proporcione los métodos para interactuar con el juego a los agentes (sensores y actuadores) [12].

# Parte IV. Implementación

En esta parte se explica cómo se han implementado los dos componentes de la plataforma: la herramienta de generación de mapas y el juego. En concreto, se realiza una revisión de los principales algoritmos que conforman cada uno de los componentes. Finalmente, se detalla la implementación de las IAs de prueba que se han creado.

## 11. Generación de mapas

La generación de los mapas se dividió en dos partes principales: la generación de la superficie y la generación de los objetos del terreno. El proceso seguido para generar la superficie esta basado en las explicaciones de Amit Patel [24]. Por su parte, la generación procedural de los objetos del terreno se llevó a cabo basándose en los artículos sobre el tema de «Jayelinda» [25].

### 11.1. Generación de la superficie

#### Diagrama de Voronoi

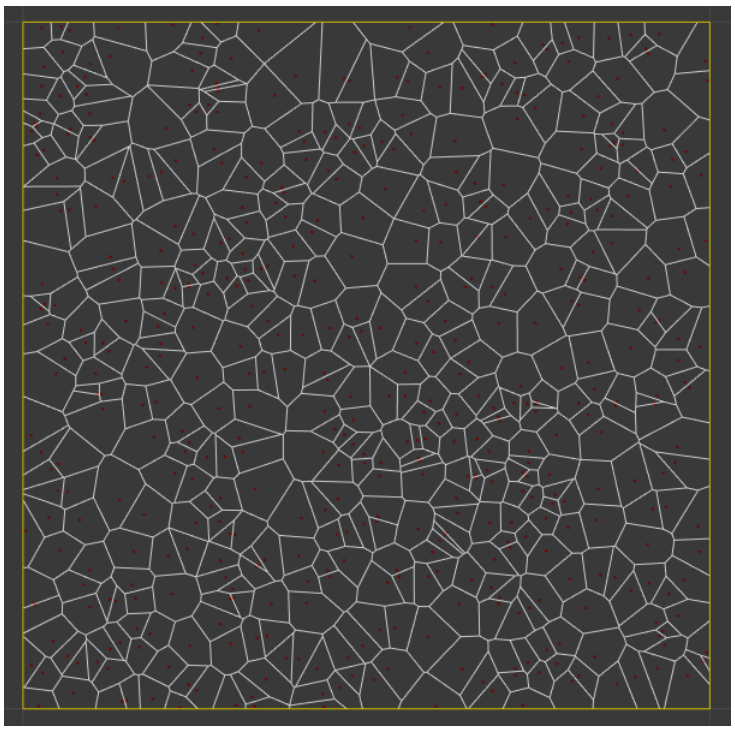


Fig. 9: Diagrama de Voronoi con 500 centros

El primer paso de la generación de la superficie consiste en realizar una partición en polígonos convexos de un plano rectangular. Una forma de hacer esto es crear el diagrama de Voronoi [27] del rectángulo. Dado un recinto rectangular y un conjunto de puntos en ese recinto, llamados centros,

su diagrama de Voronoi consiste en determinar para cada centro, el conjunto de puntos del recinto que están más próximos de ese centro que de todos los demás. Así, un diagrama de Voronoi de un rectángulo, es una partición en polígonos convexos de dicho rectángulo, cada uno asociado a un centro.

El algoritmo de Voronoi es relativamente complejo de implementar y se optó por utilizar una librería ya existente en el proyecto [28]. Sin embargo, el formato en el que dicha librería almacena los datos no era óptimo para los propósitos de este proyecto, y se implemento una capa adicional que transformaba el producto de esta librería en una estructura de grafo doble más conveniente. Dicho grafo lo forman los vértices de los polígonos del diagrama de Voronoi, asociados a los centros de los polígonos a los que pertenece cada vértice.

### Definición de tierra-agua

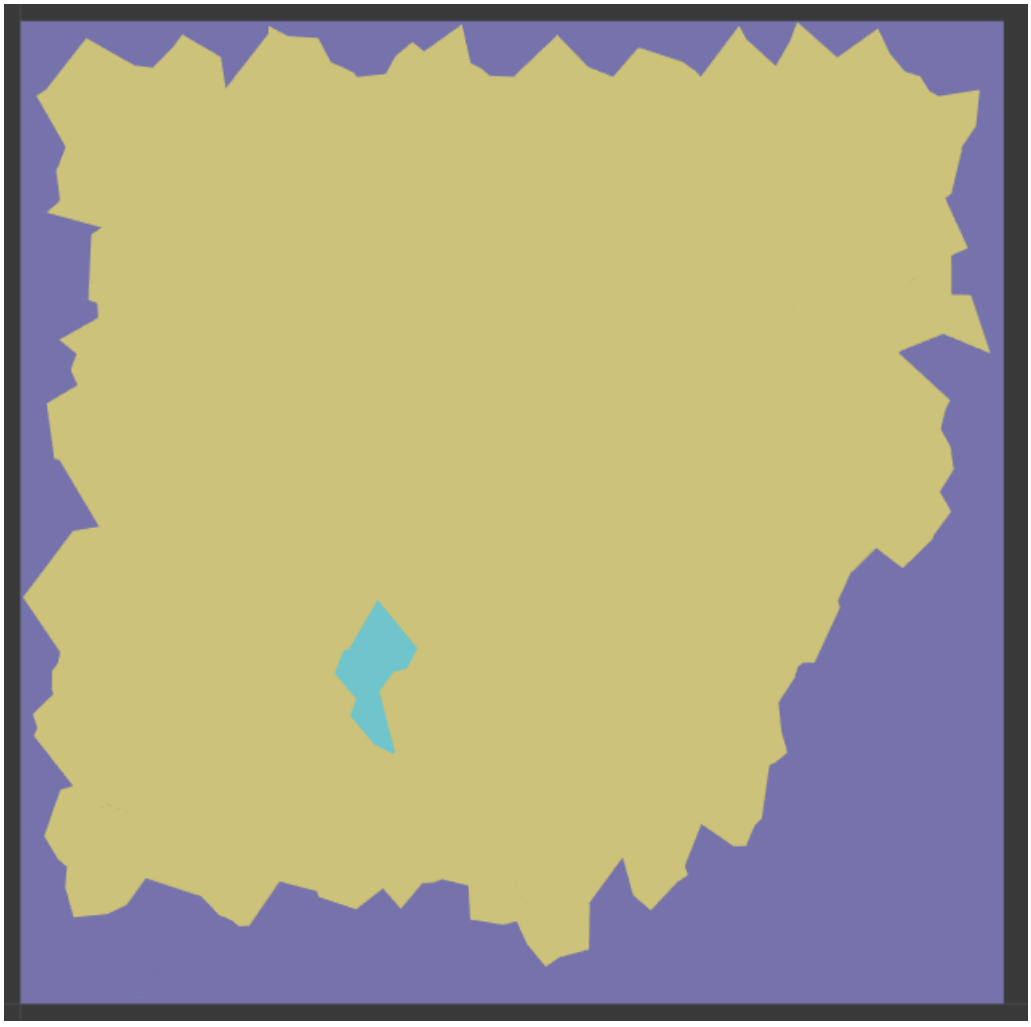


Fig. 10: Mapa con tierra y agua

Después de definir el diagrama de Voronoi del rectángulo que forma el mapa, se definen las partes que son agua y las que son tierra. Los mapas que genera la herramienta son siempre islas. Las islas se definen dividiendo el plano en sectores de un cierto ángulo respecto al centro del del rectángulo (definido como el punto de corte de las dos diagonales) y utilizando ruido rosa [29] para determinar un cierto radio pseudoaleatorio de la isla cada sector. De esta forma, todos los vértices de los polígonos de Voronoi que caigan dentro del radio de su sector se marcan como tierra y los demás como agua. Cabe destacar que se utiliza ruido rosa porque se encuentra en muchos lugares de la naturaleza y en el contexto de la generación procedural de contenido es usado a veces para definir superficies con grandes ondulaciones pero también irregularidades pequeñas.

Como estamos definiendo una isla, también se marcan como agua siempre todos los vértices de los polígonos que tengan alguna arista en el borde del rectángulo. Después de esto, se marcan como agua todos los centros cuyos polígonos tengan todos sus vértices marcados como agua y los demás centros como tierra.

Para crear los lagos, se sigue un proceso análogo al de la creación de la isla, pero en este caso seleccionando centros pseudoaleatoriamente con ruido blanco [29] dentro de la isla y definiendo la parte de dentro del radio de cada sector como agua. Por último, se marcan como océanos todos los centros marcados como agua que no sean lagos.

#### Definición de la elevación

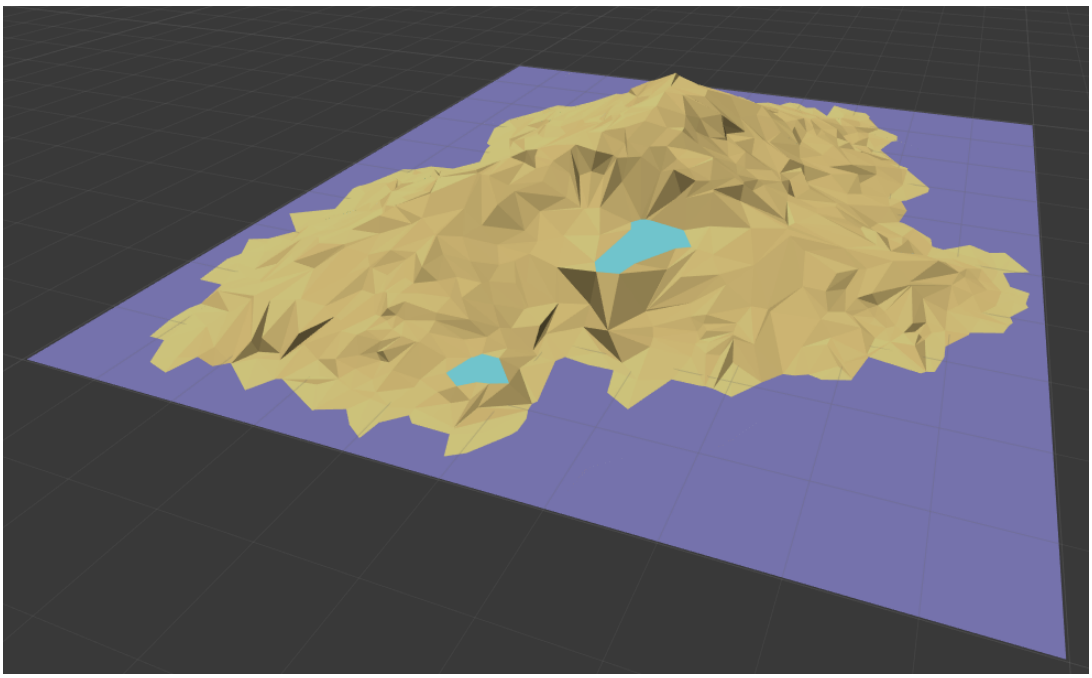


Fig. 11: Mapa con elevación (perspectiva)

Una vez definidas las zonas de agua y tierra, se procede a definir la elevación. La elevación de cada vértice de tierra de cada polígono se define como un factor de su distancia a la costa. Los

vértices de océano tienen elevación 0. Sin embargo, este proceso produce islas muy escarpadas, para solucionar esto, se realiza una redistribución de la elevación de forma que la elevación $x$ (escalada al intervalo $[0, 1]$) tenga frecuencia $1 - x$. Para ello, se ordenan los vértices en función de su elevación y se ajusta la elevación para conseguir la distribución deseada.

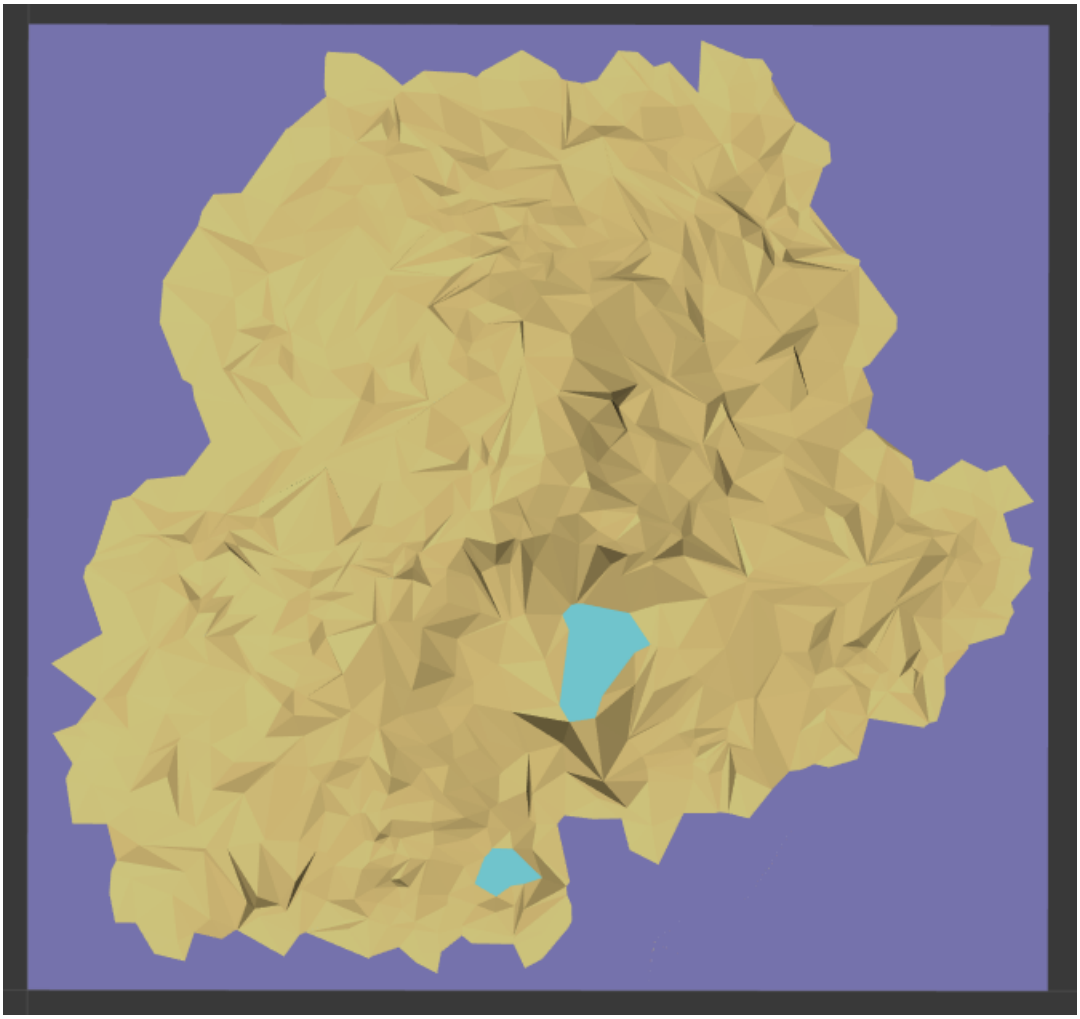


Fig. 12: Mapa con elevación (top-down)

El último paso consiste en ajusta la elevación de los lagos, que deben ser planos. La elevación de los vértices de cada lago se define como la media aritmética de las elevaciones de todos los vértices del lago.

### Definición de ríos

Aparte de ser un método sencillo de modelar la elevación de una isla, definir la elevación de forma directamente proporcional a la distancia a la costa en cada vértice sirve un doble propósito, pues facilita que dado un vértice en la isla, exista un camino por las aristas que llegue hasta el mar

o hasta un lago, y es aprovechando esta característica como se construyen los ríos.

Se seleccionan vértices aleatorios usando ruido blanco a partir de una cierta elevación mínima como los orígenes de los ríos. A partir de ahí, se recorre el grafo desde cada origen tomando en las intersecciones la arista de mayor pendiente y aumentando el volumen de agua que lleva el río en ese punto cada vez por cada río que pase por él. Los ríos acaban cuando llegan a la costa o cuando llegan a un vértice degenerado por la redistribución de elevaciones en el que no exista una pendiente máxima bien definida.

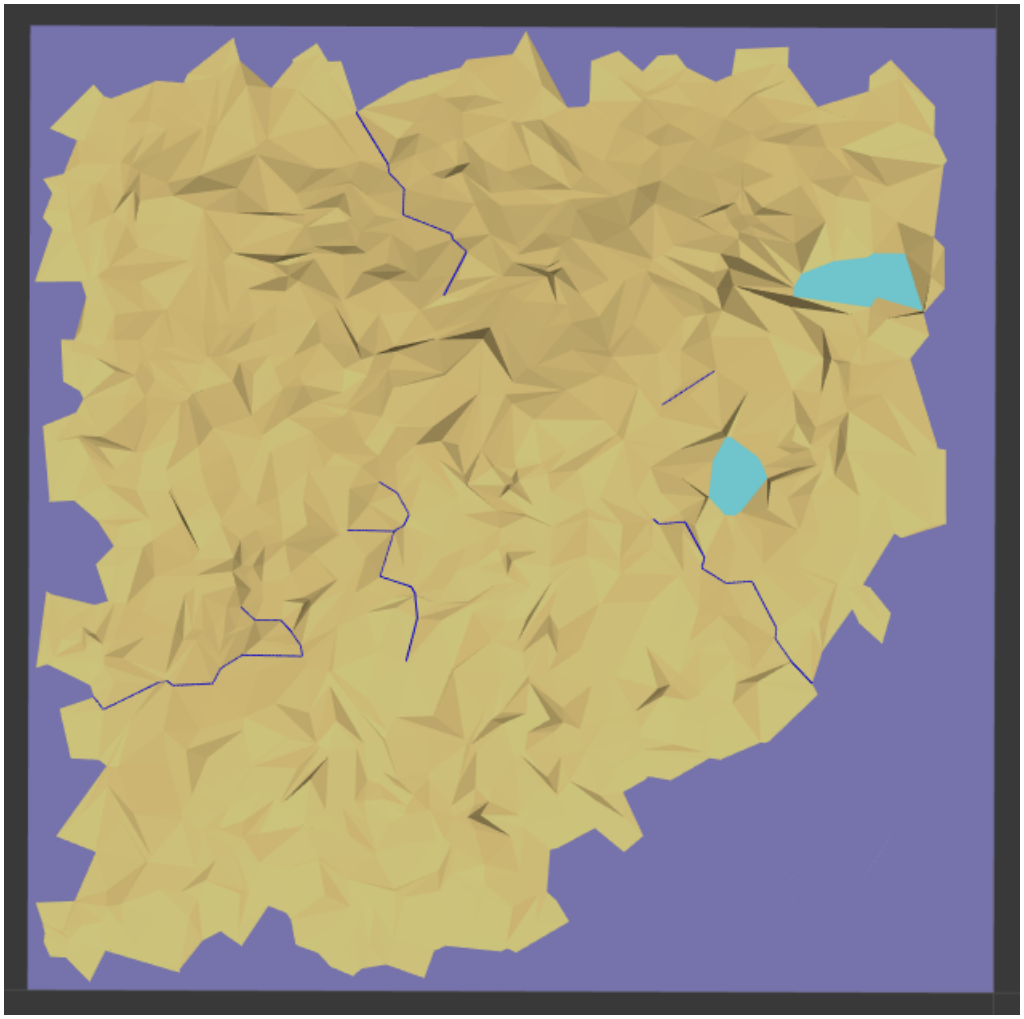


Fig. 13: Mapa con ríos

### Definición de humedad

En este punto, se define la humedad en cada uno de los vértices propagándola desde las fuentes de agua dulce (ríos y lagos). Se forma una cola con todos los vértices por los que pase un río o que sean parte de un lago. Después se van sacando vértices de la cola y propagando la humedad reducida por un cierto factor a los vértices adyacentes, que se meten en la cola la primera vez que se

aumenta su humedad también y se procede hasta que la cola queda vacía. Después sólo resta definir la humedad de los centros como la media de la humedad de los vértices de sus polígonos asociados.

### Definición de biomas

Habiendo definido la elevación y la humedad de cada centro/polígono, tenemos un buen criterio para definir biomas, para la que se usa la clasificación que propone en su artículo Amit Padel [24], que a su vez es una versión adaptada del diagrama de biomas de Whittaker [30].

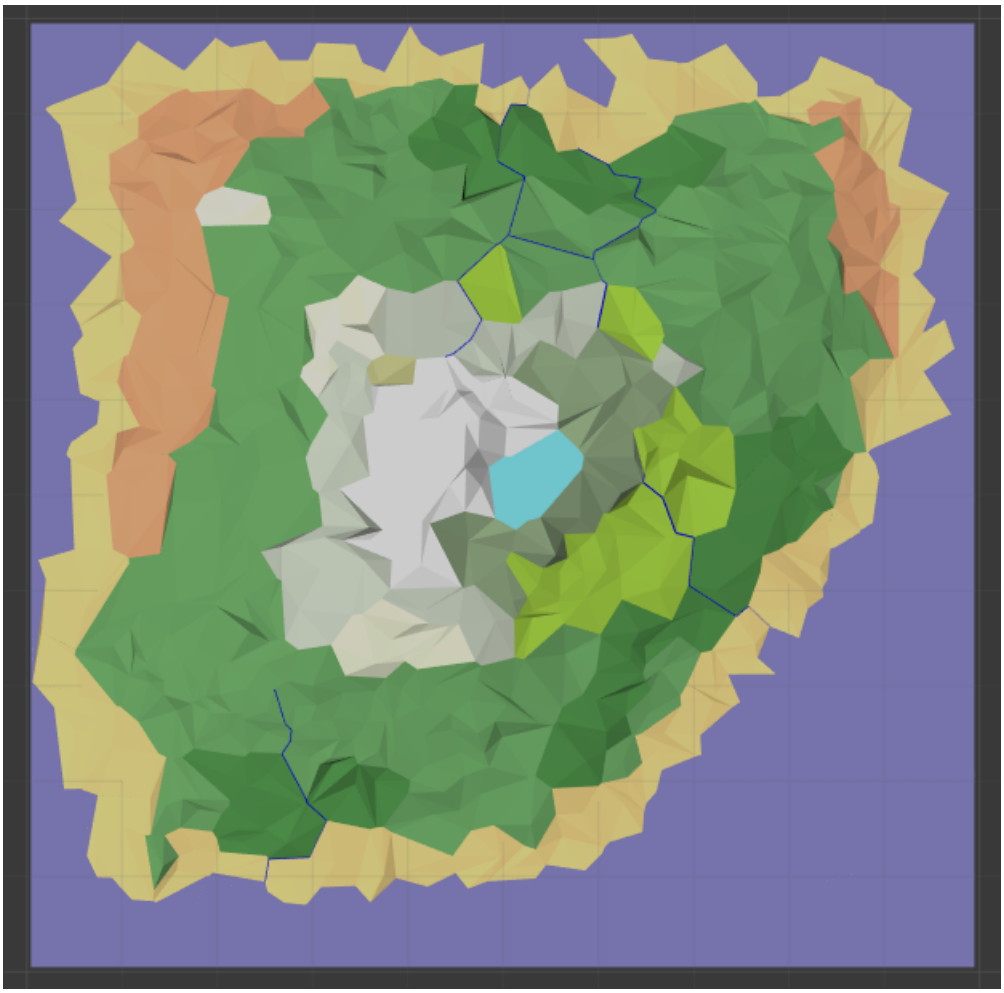


Fig. 14: Mapa con biomas

La Tabla 6 indica la correspondencia de colores y biomas en el mapa.

| Elevación | Humedad | | | | | |
|---|---|---|---|---|---|---|
| | 6 | 5 | 4 | 3 | 2 | 1 |
| 4 | Snow | | | Tundra | Bare | Scorched |
| 3 | Taiga | | Shrubland | | Temperate Desert | |
| 2 | Temperate Rain Forest | Temperate Deciduous Forest | | Grassland | | Temperate Desert |
| 1 | Tropical Rain Forest | | Tropical Seasonal Forest | | Grassland | Subtropical Desert |

| |
|---|
| Playa |
| Lago helado |
| Lago |
| Oceano |

Tab. 6: Correspondencia colores-biomas

Una gran cantidad de aspectos del proceso de generación de superficies están parametrizados y los valores de dichos parámetros pueden ser modificados por el usuario a través del editor de *Unity3D* para adecuar los resultados a sus necesidades. La Figura 15 muestra los parámetros que se ofrecen para ajustar.

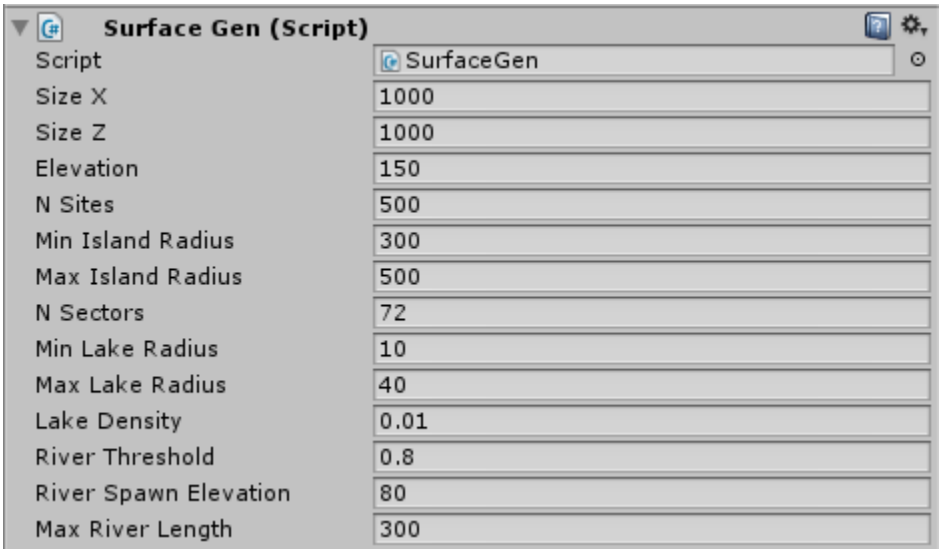



Fig. 15: Parámetros del generador de superficies

## 11.2. Generación de los objetos del terreno

La generación de los objetos del terreno tiene dos partes principales: la construcción de los objetos y la distribución de los objetos en el terreno.

### Construcción de objetos del terreno

La Figura 16 muestra un ejemplo de cada uno de los tipos de objeto del terreno que se pueden construir, a saber: árboles, arbustos y troncos caídos.

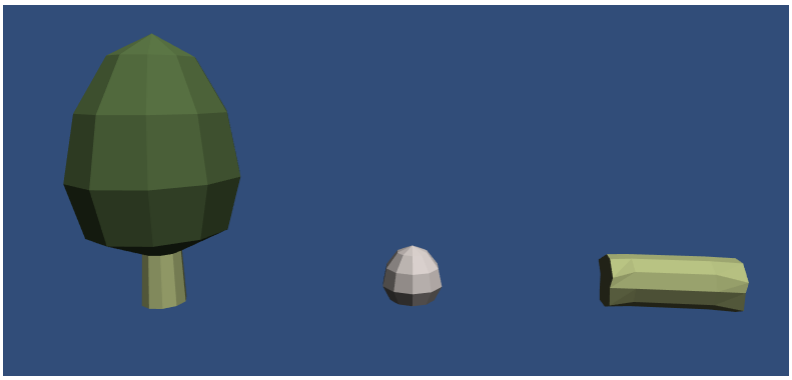


Fig. 16: Objetos del terreno

La Figura 17 muestra los parámetros de generación de los árboles y parámetros análogos definen la generación de los arbustos y los árboles. Eligiendo de forma aleatoria estos parámetros se obtiene una cierta variedad de árboles, arbustos y troncos sencillos.

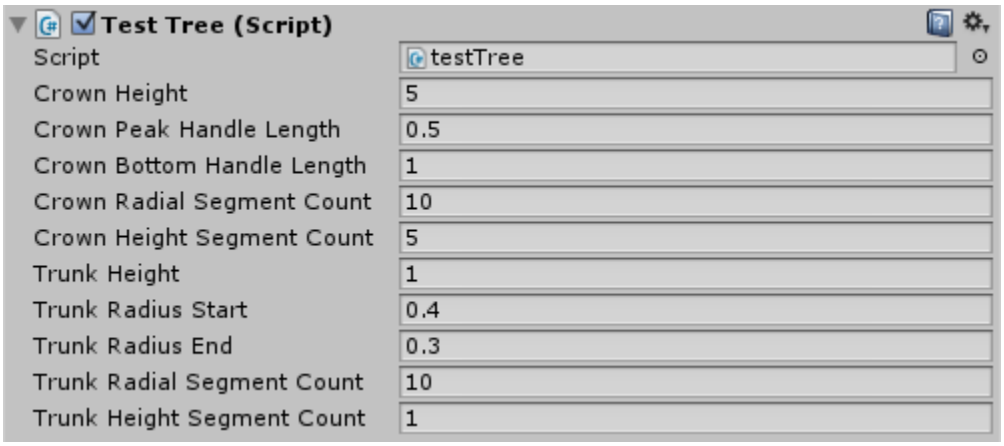



Fig. 17: Parámetros de configuración de los árboles

La construcción de todos los objetos se basa en variaciones de una misma técnica. A excepción de unas pequeñas perturbaciones en los extremos de los troncos caídos, los tres objetos se forman uniendo anillos de vértices a distintas alturas y con diferentes radios. Se resume a continuación el proceso de construcción de los objetos.

En el caso de los árboles, el tronco se construye uniendo tantos anillos como determinen los parámetros y el radio de los anillos se define realizando una interpolación lineal en función de la altura a la que esté el anillo desde el radio en la base del tronco hasta el radio en la parte superior del tronco. Después, partiendo del último anillo del tronco, el radio de los anillos de la copa se define la curva de *Bézier* [31] de asas definidas por los parámetros `CrownBottomHandleLength` y `CrownPeakHandleLength`.

Los arbustos son sencillamente árboles sin tronco y los troncos caídos se construyen como los troncos de los árboles, pero añadiendo una desviación aleatoria a los vértices de los extremos, cerran-

do los anillos de los extremos en un punto central situado a una distancia también parametrizada del anillo en la dirección normal del plano del anillo y rotando toda la estructura para que el tronco se apoye de costado en el suelo.

### Distribución de los objetos del terreno[1]

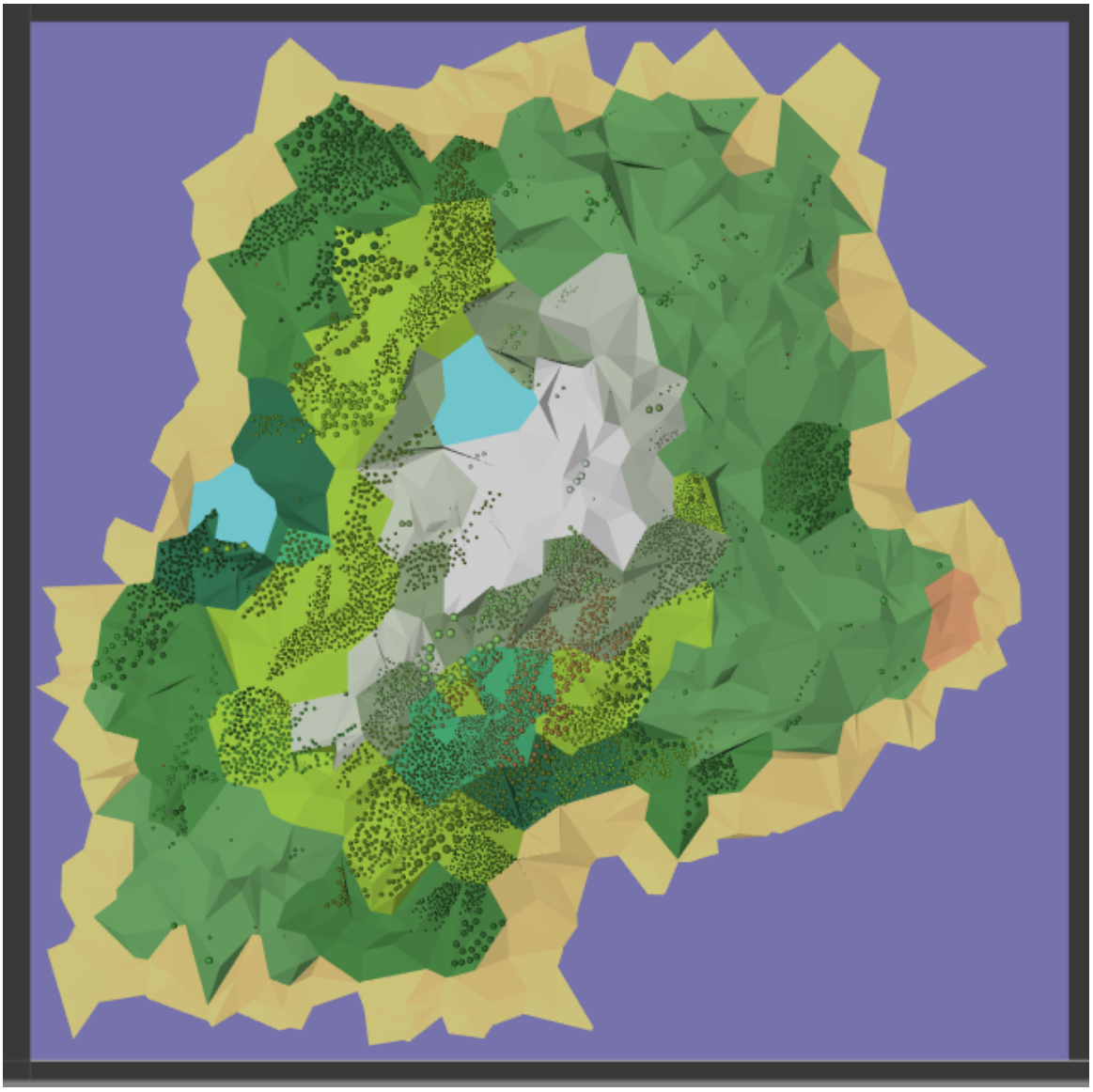


Fig. 18: Mapa con objetos del terreno (top-down)

El siguiente paso es la distribución de los objetos en el terreno. En esta parte se utilizan los biomas definidos en el generador de superficies para definir la probabilidad de que haya un árbol o

[1] Tree icon by Valentina Piccione from the Noun Project

un arbusto en un determinado punto del mapa. El primer paso es seleccionar un conjunto de puntos aleatoriamente en el mapa, y con una probabilidad definida en función del bioma en el que esté el punto, decidir si en cada uno de los puntos hay un objeto. En el caso de los troncos caídos, este es el método seguido. Sin embargo, en el caso de los árboles y los arbustos, el proceso es algo más elaborado para lograr las distribuciones más interesantes que se aprecian en la Figura 18.

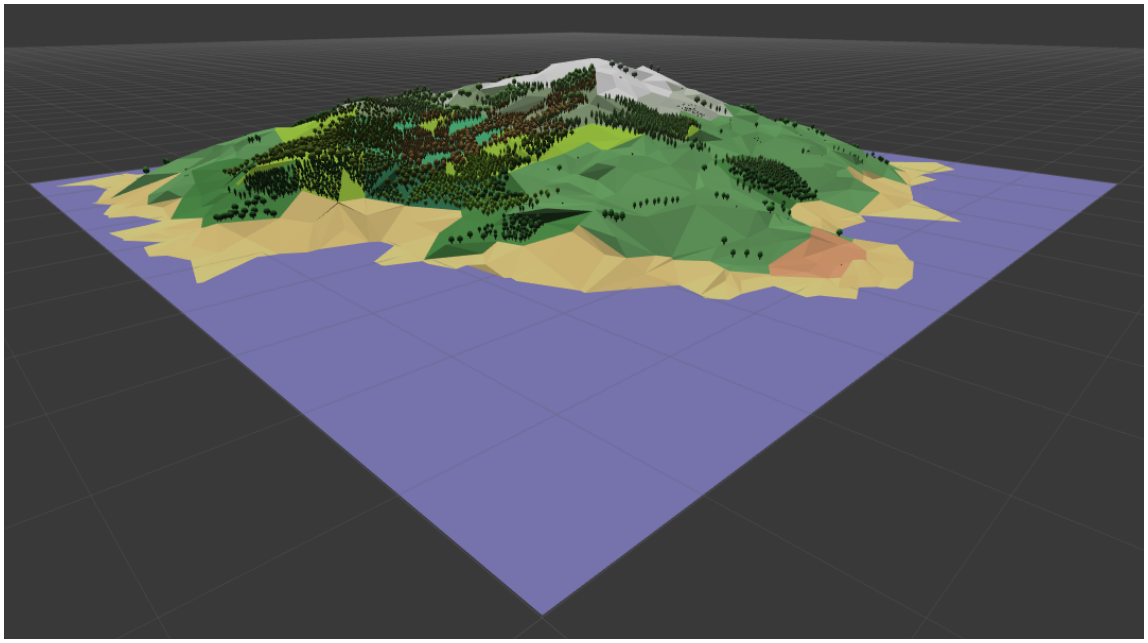


Fig. 19: Mapa con objetos del terreno (perspectiva 1)

Se empieza por crear una primera generación de árboles siguiendo el método que se acaba de describir. Después de esto, cada árbol deja un número aleatorio de semillas en un anillo a su alrededor. Las semillas se convierten en árboles con una probabilidad que es función decreciente del radio y proporcional al parámetro de probabilidad de árboles o arbustos del bioma del punto estudiado, como muestra la Figura 20. Este proceso se repite con las nuevas generaciones de árboles producidas por semillas también.


Fig. 20: Diagrama del sistema de semillas para la distribución de árboles y arbustos

Este método sigue la lógica siguiente. En la proximidad inmediata de la planta padre, los hijos no sobreviven porque hay competencia por recursos (nutrientes, agua y luz), sin embargo, cuanto más lejos de la planta padre, más improbable es que las semillas llegen arrastradas por el viento, de modo que una vez la competencia por recursos con el padre se vuelve despreciable, la probabilidad de que una semilla llegue a un punto es inversamente proporcional a la distancia al padre. Este método de distribución de los árboles y arbustos produce formaciones boscosas, que es justo lo que se pretendía.

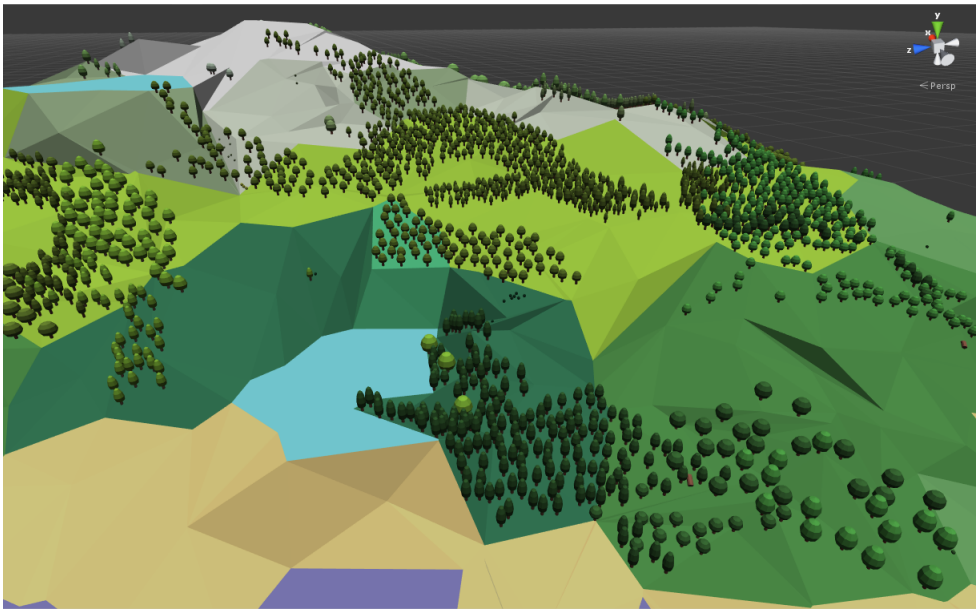


Fig. 21: Mapa con objetos del terreno (perspectiva 2)

Un último aspecto del sistema de semillas es que los árboles producidos por semillas tienen el

mismo aspecto que sus árboles padres y que el aspecto de los árboles de primera generación está determinado por parámetros de valores aleatorios en la forma y por el bioma en el color.

En la Figura 22 se muestran los parámetros que ofrece el generador de objetos al usuario y que permiten ajustar los resultados producidos en función de sus necesidades.


Fig. 22: Parámetros del generador de objetos del terreno

## 11.3. Interfaz gráfica de usuario

Para la creación de la interfaz gráfica, se ha usado el sistema para crear interfaces gráficas de *Unity3D* [35].

En la Figura 23 se muestra la interfaz gráfica de usuario de la herramienta de generación de mapas. Pulsando los botones *Generate Surface* y *Generate Terrain Objects* se generan nuevas superficies y nuevos objetos del terreno respectivamente. Para poder generar objetos del terreno, hay que generar primero la superficie, y si una vez generados los objetos del terreno se regenera la superficie, se eliminarán los objetos creados.

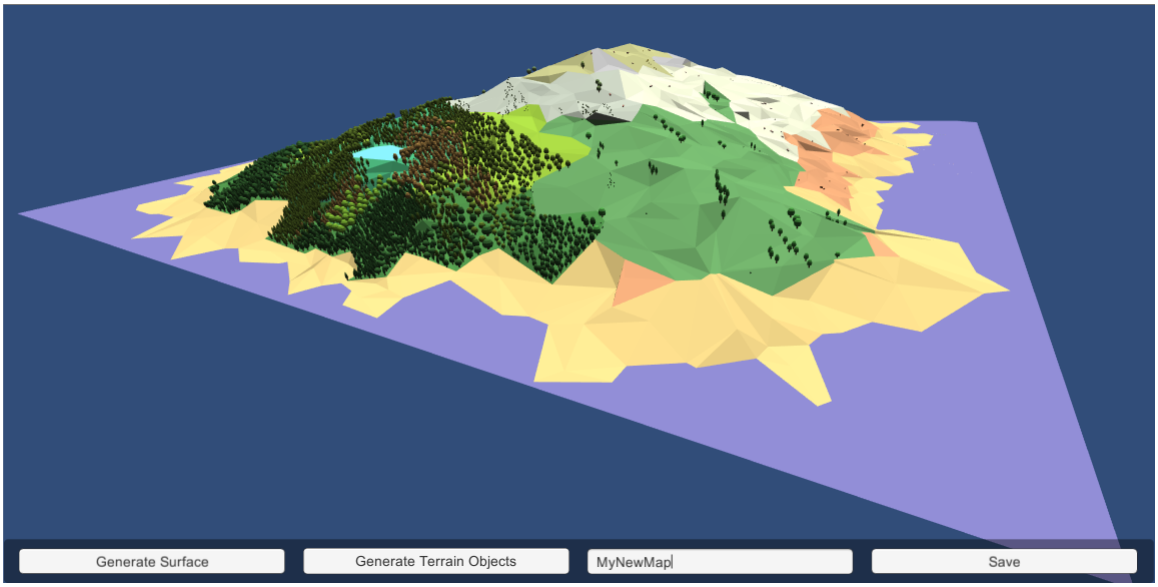



Fig. 23: Interfaz gráfica de la herramienta de generación de mapas

Por otro lado, pulsando el botón *Save* después de haber introducido un nombre para el mapa en el *input field* mostrado en la figura, se guarda una escena de *Unity3D* con el mapa.

En la Figura 24 se muestran los mensajes que muestra la aplicación a través de la consola del editor de *Unity3D* en cada uno de los pasos mencionados.

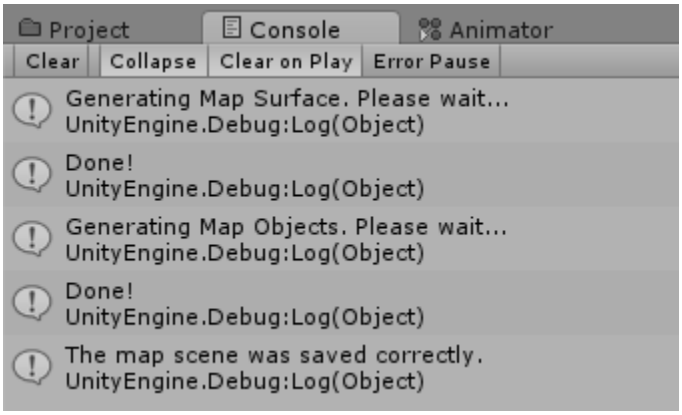



Fig. 24: *Feedback* de la herramienta de generación a través del editor de Unity3D

# 12. Juego

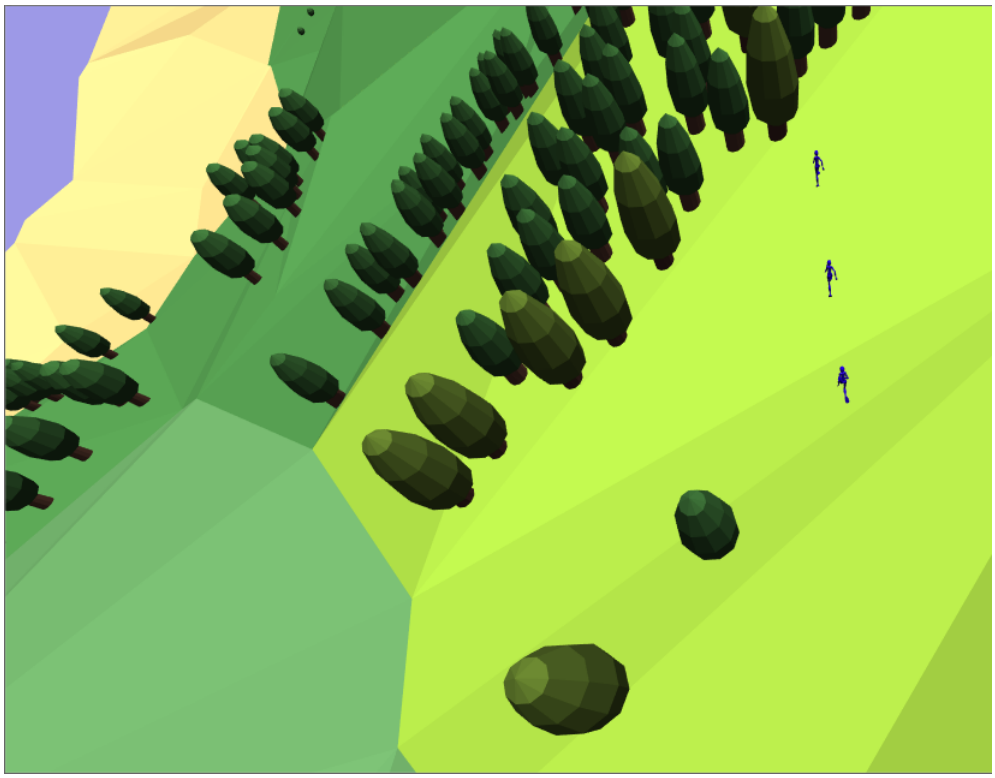


Fig. 25: Tres agentes HeuristicAI siguiéndose entre ellos

## 12.1. Gestión de turnos

Una vez empezada una partida, se sigue un ciclo alternando la recolección de información del entorno para los jugadores, y la ejecución de los agentes. La recolección de la información para los agentes es un proceso demasiado largo para ser ejecutado en un único *frame*, que dura 0.02 segundos por defecto. Para solucionar esto, se usan las corutinas de *Unity3D* [22], que permiten interrumpir la ejecución de una función, guardando el estado de su entorno y programar su reanudación en el futuro. En este caso, se utiliza la clase `System.Diagnostics.StopWatch`, en conjunción con medidas estadísticas experimentales de la duración de las distintas fases de recolección de la información para controlar el tiempo de ejecución e interrumpir la ejecución cada 0.02s y reanudarla en el siguiente *frame*.

Una vez se ha recolectado la información del entorno para los jugadores, se les da acceso a ella y se procede a darles secuencialmente el control a los agentes durante un *frame* para que tomen decisiones. Como se mencionó en el apartado de diseño, los propios agentes deben realizar un proceso semejante al descrito en el párrafo anterior para controlar el tiempo de ejecución, aunque, de momento, la ejecución de los agentes debe ser atómica.

Con este sistema se completan dos ciclos por segundo aproximadamente, es decir, cada jugador

tiene dos «turnos» cada segundo.

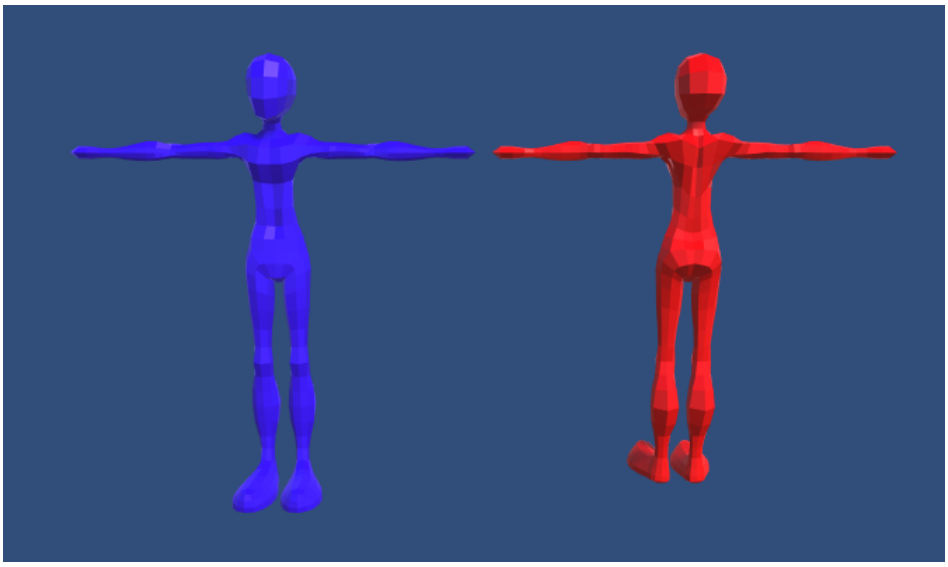


Fig. 26: Avatares azul y rojo

## 12.2. Sensores: Oído

Cada vez que un jugador realiza una acción que produce ruido, se crea una instancia de la clase `NoiseSource`. Durante la recolección de información para los jugadores, como se muestra en la Figura 27[2] se utiliza la distancia de los jugadores a la fuente de sonido (`NoiseSource`) para determinar qué jugadores oyen el ruido y a estos jugadores se les da acceso a una instancia de la clase `Noise`, que contiene información acerca de la dirección en la que estaba la fuente del sonido en el momento de producirse y la acción que causo el ruido. Cabe destacar que las diferentes acciones producen ruidos con distintos alcances.

En el caso particular de que el origen del ruido fuese una acción de comunicación, la instancia de la clase `Noise` contiene también la información que se comunicase.

[2] Person icon by Ferran Brown from the Noun Project, Speaker icon by Harold Kim from the Noun Project

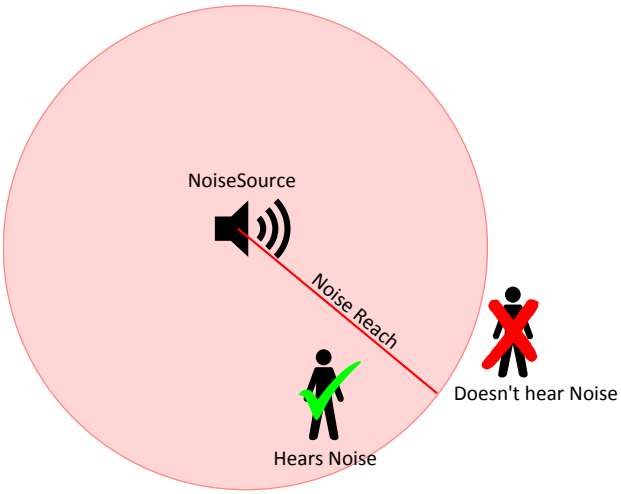



Fig. 27: Sistema de oído

## 12.3. Sensores: Vista

El área de visión de cada jugador se define como el sector de circunferencia de un cierto radio y ángulo con centro en la posición del jugador, orientado en la dirección en le que mira el jugador, como se muestra en la Figura 28[3]. Para determinar si un jugador ve a un cierto objetivo (otro jugador o un objeto del terreno) se comprueba primero si el objetivo está dentro de área de visión del jugador observador. Después, para simular la obstrucción de la visión producida por el relieve, otros jugadores u otros objetos del terreno, se utilizan los *Colliders* [32] del mapa creados por la herramienta de generación y una serie de *Raycasts* [34, 33] entre el observador y el objetivo para determinar si la visión está obstruida o no.

Cuando un jugador es visto por otro, se da acceso al observador a una instancia de la clase `SightedActor`, que contiene información acerca de la posición del jugador avistado, la dirección en la que está mirando y qué acción está realizando en ese momento el jugador avistado. Cuando un jugador ve un objeto del terreno, se le da acceso a información acerca de la posición y las dimensiones del objeto.

[3] Check Mark icon by Danny Sturgess from the Noun Project, Cross icon by Spencer Harrison from the Noun Project, Tree icon by Valentina Piccione from the Noun Project

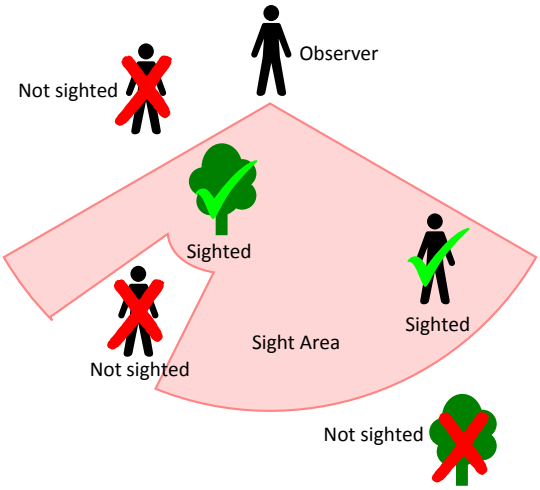



Fig. 28: Sistema de visión

## 12.4. Comunicación

Los jugadores pueden comunicar información a otros jugadores, creando instancias de las mismas clases que el juego utiliza para proporcionar información sobre su entorno a los jugadores. Esto restringe las posibilidades de la comunicación, pero al mismo tiempo garantiza que ambos equipos podrán entender la información comunicada, lo cual es importante, pues la intercepción de información enemiga es una de las mecánicas del juego.

## 12.5. Disparos

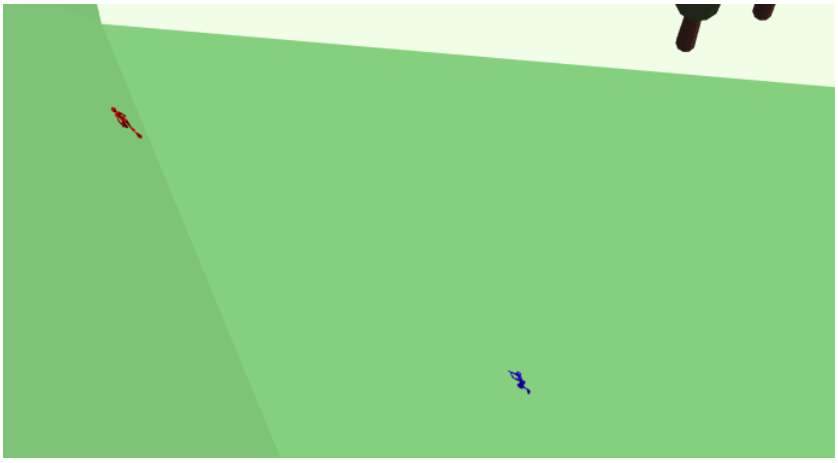


Fig. 29: Jugador persiguiendo y apuntando a un adversario

El proceso para determinar si un jugador consigue impactar a su objetivo al disparar consiste en lanzar un *Raycast* [34, 33] desde el jugador que dispara hasta la parte del cuerpo del objetivo a la que

apunta para comprobar si el disparo choca con algún obstáculo antes del alcanzar al objetivo. Sin embargo, se impone un desvío aleatorio sobre la trayectoria del disparo para modelar la dificultad del disparo. La magnitud del desvío depende de la postura y el movimiento del jugador que dispara y del jugador objetivo y el tiempo que se ha invertido en apuntar. De forma indirecta, la parte del cuerpo a la que se ha apuntado también influye en la probabilidad de acertar, pues un desvío menor provoca que la trayectoria nunca alcance al objetivo.

## 12.6. API Actor

La clase `Actor` representa a los personajes dentro del juego y sirve de interfaz entre los agentes implementados por los usuarios y el juego, esto es, da acceso a los jugadores a la información que les proporciona el juego acerca de su entorno y también les ofrece una serie de métodos con los que pueden hacer que su avatar dentro del juego realice acciones. La Figura muestra las acciones y otros métodos para obtener información del estado del jugador que expone la clase `Actor`, así como la variable que el juego actualiza automática y periódicamente con información actualizada del entorno del jugador.

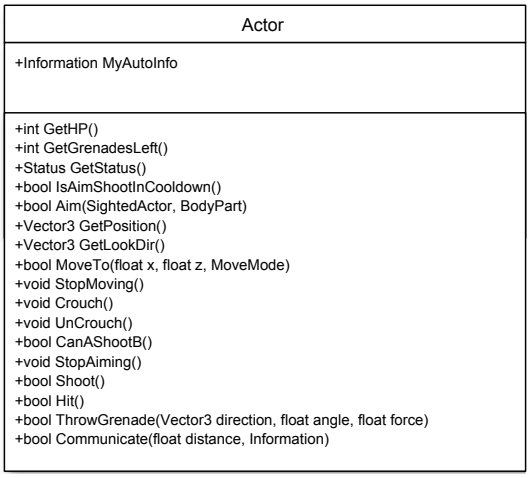



Fig. 30: Variables y métodos expuestos por la clase `Actor`

## 12.7. Sistema de logs

Se ha implementado un sistema de registro de los sucesos de las partidas. El sistema registra las acciones, posiciones y estados de los jugadores con una marca temporal y serializa en formato *XML,* utilizando la librería `System.Xml.Serialization`, cada cierto intervalo de tiempo.

## 12.8. Sistema de puntuación

El sistema de puntuación registra los siguientes parámetros:

- **Vidas quitadas**, que representa el número de veces que se ha alcanzado a enemigos con un ataque cualquiera.
- **Cuerpo a cuerpo**, que representa el número de veces que se ha alcanzado a enemigos con un ataque cuerpo a cuerpo.
- **Disparando**, que representa el número de veces que se ha alcanzado a enemigos con un disparo.
- **Granadas**, que representa el número de veces que se ha alcanzado a enemigos con una granada.
- **Con Sigilo**, que representa el número de veces que se ha alcanzado a enemigos con un ataque antes de que el enemigo viese al jugador.
- **A la cabeza**, que representa que representa el número de veces que se ha alcanzado a enemigos con un disparo dirigido a la cabeza al apuntar.
- **Muertes múltiples**, que representa el número de veces que se ha conseguido alcanzar a más de un enemigo con un mismo ataque.
- **Fuego amigo**, que representa el número de veces que se ha alcanzado a aliados con un ataque.
- **Vidas restantes**, que representa el número de puntos de vida que conserva un jugador.
- **Granadas restantes**, que representa el número de granadas que que conserva un jugador.

Fig. 31: Jugador deshabilitado tras ser alcanzado por un ataque

La puntuación por cada vida quitada se computa de acuerdo a las siguientes fórmulas:

- $Puntuación = friendlyFire * unseen * headShot * multiKill$
- $friendlyFire = \begin{cases} -1 & si\,se\,alcanzó\,a\,un\,aliado \\ 1 & si\,se\,alcanzó\,a\,un\,enemigo \end{cases}$
- $unseen = \begin{cases} 2 & si\,fue Con\,Sigilo \\ 1 & en\,otro\,caso \end{cases}$
- $headShot = \begin{cases} 2 & si\,fue\,un\,disparo\,a\,la\,cabeza \\ 1 & si\,no \end{cases}$
- $multiKill = \begin{cases} 2 * \#enemigosAlcanzados^2 & si\,\#enemigosAlcanzados > 1 \\ 1 & en\,otro\,caso \end{cases}$

Usando estas fórmulas y parámetros, se definen los siguientes:

- **Puntos por vidas quitadas/muertes** es la suma de las puntuaciones por cada vida quitada por el jugador.
- **Bonus por vida restante**$= \begin{cases} 2 & si\,Vidas\,Restantes = 5 \\ 1 & si\,Vidas\,Restantes > 0 \\ 0 & si\,Vidas\,Restantes = 0 \end{cases}$
- **Bonus por granadas restantes**$= \begin{cases} 2 & si\,Granadas\,restantes > 0 \\ 0 & en\,otro\,caso \end{cases}$
- **Bonus por puntería**$= \begin{cases} 2 & si\ \frac{Disparando}{\#disparosRealizados} >= 0{,}9 \\ -2 & si\ \frac{Disparando}{\#disparosRealizados} <= 0{,}1 \\ 0 & en\,otro\,caso \end{cases}$
- **Bonus por variedad**$=$ mín $(Cuerpo\,a\,cuerpo, Disparando, Granadas)$

Finalmente, utiliza estos parámetros para computar la puntuación total de cada jugador como:

$$\begin{aligned} Puntos\,totales \quad = \quad & Puntos\,por\,vidas\,quitadas + Bonus\,por\,vida\,restante + \\ & Bonus\,por\,granadas\,restantes + Bonus\,por\,puntería + Bonus\,por\,variedad \end{aligned}$$

Y la puntuación total de cada equipo como la suma de las puntuaciones totales de cada jugador del equipo.

## 12.9. Interfaz gráfica de usuario

Para la creación de la interfaz gráfica, se ha usado el sistema para crear interfaces gráficas de *Unity3D* [35].

La Figura 32 muestra la interfaz gráfica del juego y su navegación mediante teclado y ratón. Durante las partidas, se puede cambiar el jugador al que sigue la cámara usando la tecla *Tabulador* y se puede regular la distancia de la cámara con la rueda del ratón.

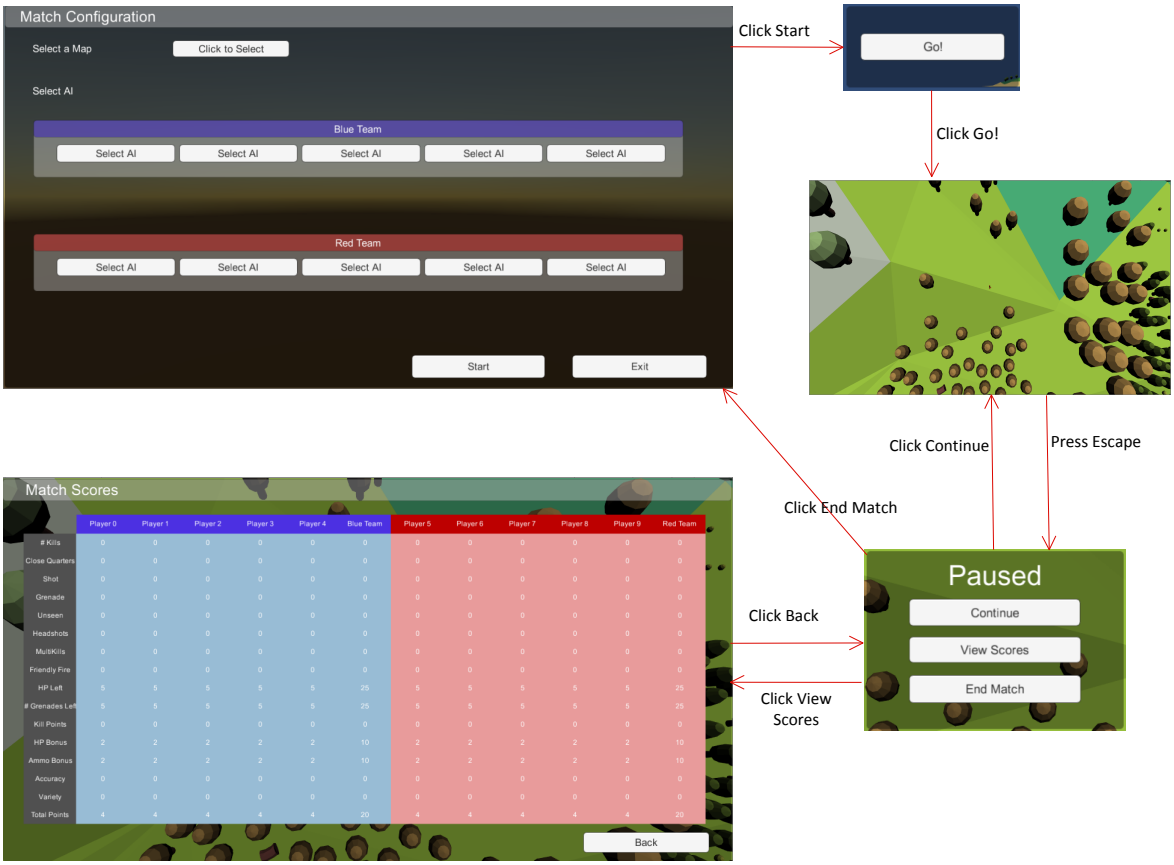



Fig. 32: Interfaz gráfica del juego y navegación

# 13. IAs de prueba

Se han implementado tres agentes de prueba para demostrar el funcionamiento de la plataforma. A continuación, se explica el funcionamiento de cada uno de ellos.

## 13.1. RandomAI

A modo de referencia básica, se ha implementado un primer jugador que toma decisiones aleatorias. Sin embargo, dada la naturaleza continua [12] del juego, se ha optado por no realizar una implementación estricta de los que sería un jugador aleatorio. La Figura 20 muestra los árboles de

decisión del jugador `RandomAI`, donde *rnd1* y *rnd2* denotan dos valores aleatorios generados cada turno.

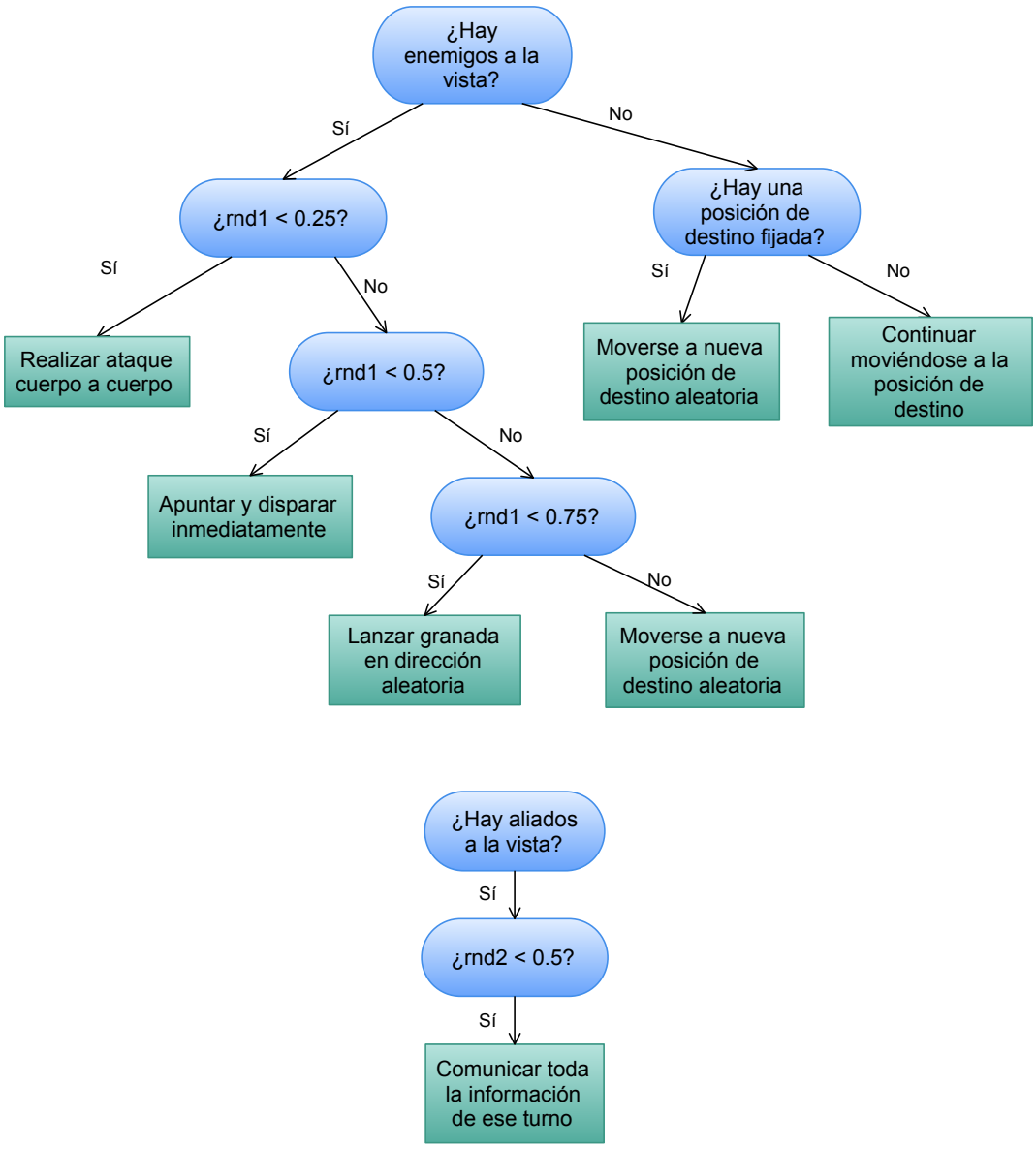



Fig. 33: Árboles de decisión de `RandomAI`

## 13.2. BasicAI

El segundo jugador que se ha implementado sigue un sistema de reglas básico para tomar decisiones, no usa la comunicación y prácticamente no analiza los ruidos que oye. La Figura 34 muestra su grafo de decisión. El agente utiliza el mismo algoritmo que el juego utiliza para calcular la magnitud del desvío sobre la trayectoria de los disparos para estimar la probabilidad de acertar.

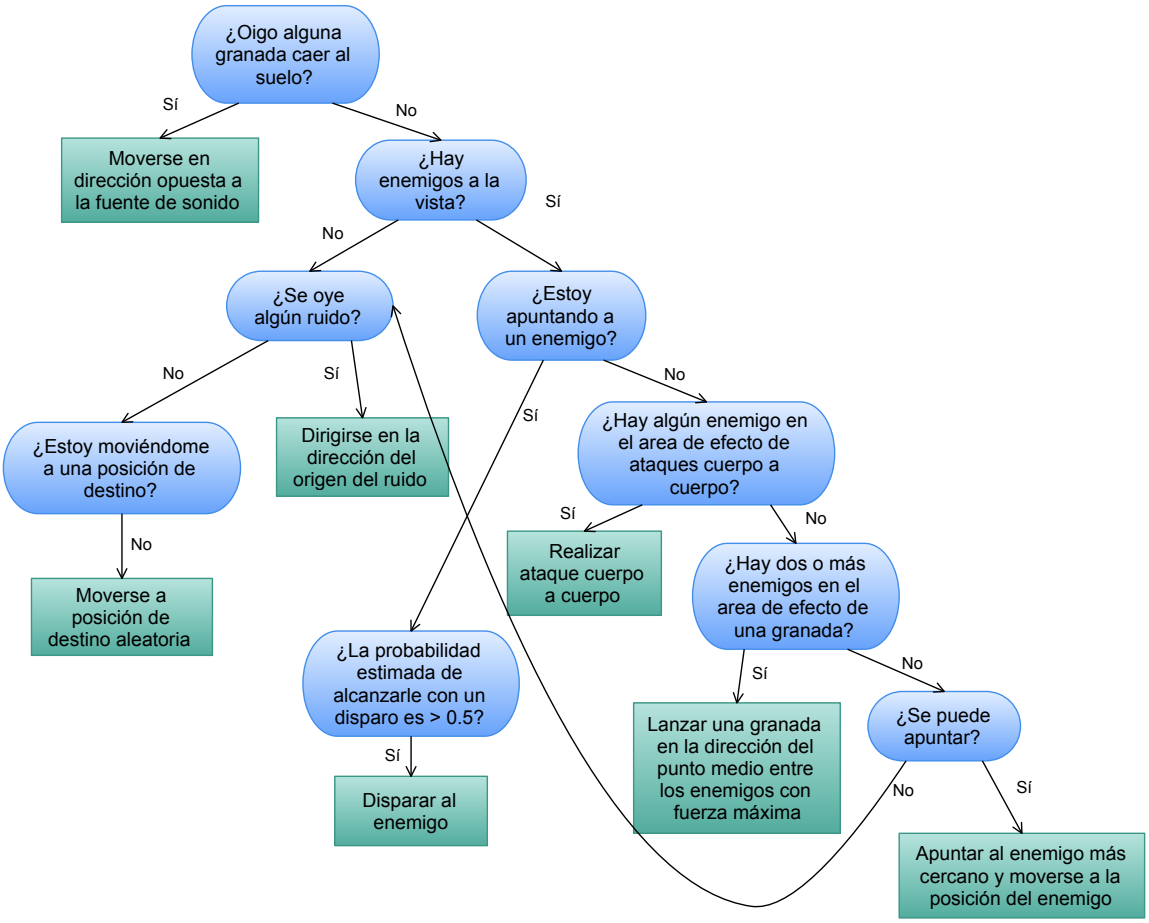



Fig. 34: Grafo de decisión de `BasicAI`

## 13.3. HeuristicAI

El tercer agente implementado hace uso combinado de un *behaviour tree* (Figura 35) y dos submáquinas de estados (Figura 36) para estructurar su comportamiento. La estrategia que sigue `HeuristicAI` es buscar a otros aliados y seguirles. Para poder usar el mismo algoritmo en los cinco jugadores del equipo, los agentes utilizar sus identificadores en el juego para establecer una jerarquía. De este modo, cada jugador sólo sigue a los jugadores de mayor rango que él. Los aliados de mayor rango que cada jugador se denominan *líderes* en las Figuras 35 y 36. Siguiendo a otros jugadores aumenta la probabilidad de que los encuentros con jugadores enemigos se den en situación de superioridad numérica a favor del «equipo `HeuristicAI`», lo que debería ser suficiente para ganar a las otras dos *IA*s de ejemplo implementadas, que es el objetivo de esta.

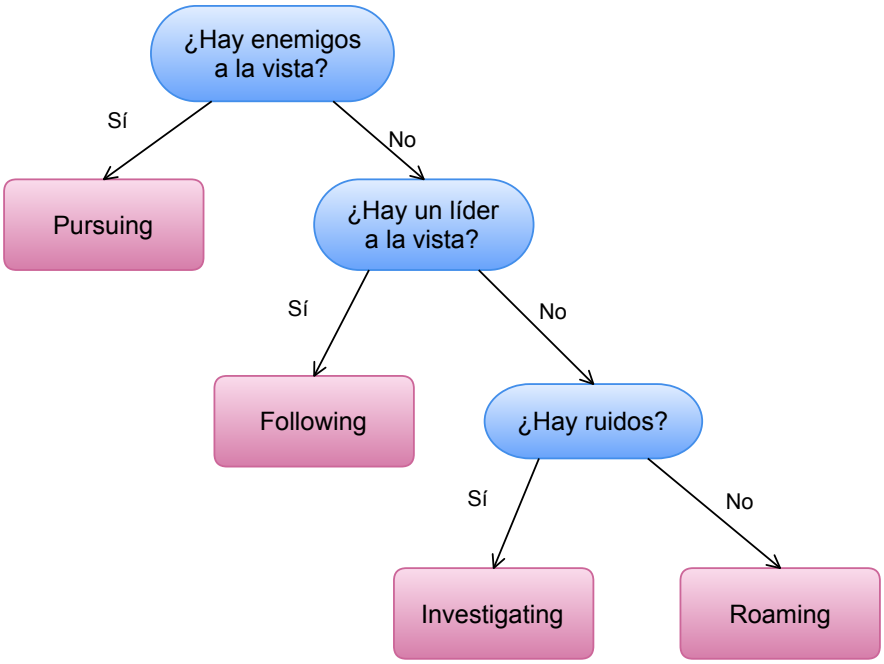



Fig. 35: Behaviour tree de `HeuristicAI`

`HeuristicAI` implementa también un método algo mejor para perseguir a sus oponentes y seguir a los *líderes*. La mejora consiste precisamente en la incorporación de los estados `LostLeader` y `LostTarget`. Al entrar en estos estados, se proyecta en el futuro la posición del *líder* o adversario seguido utilizando la dirección en la que estaba mirando en el último momento en el que se le veía. Esta mejora hace que los cambios de dirección bruscos del jugador seguido y los entornos con muchas obstrucciones a la visión provoquen muchas menos pérdidas del *líder* o del adversario, esto se traduce en más tiempo para apuntar al enemigo perseguido y refuerza la estrategia de seguir a los aliados.

Por otro lado, `HeuristicAI` realiza un análisis mejor, aunque sencillo de los ruidos que oye el jugador. Clasifica según unos criterios de prioridad los ruidos que oye e investiga los más prioritarios siempre. Además, `HeuristicAI` aprovecha el sistema de comunicación para avisar a sus aliados cuando avista a un enemigo, pero sin alertarle, y también, cuando un jugador avista a un aliado de menor rango - al que por tanto no debe seguir - le avisa de su presencia para intentar que el otro le siga.

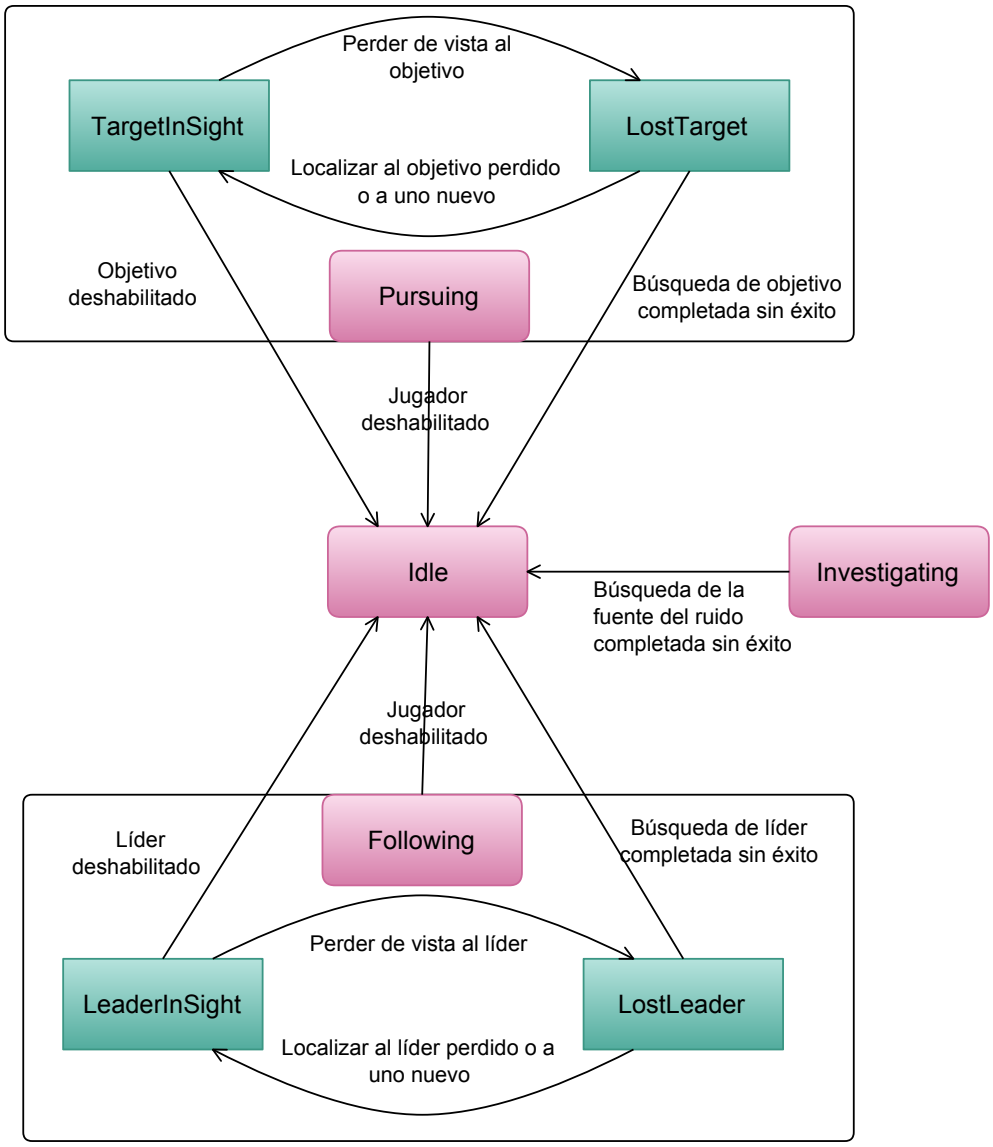



Fig. 36: Máquina de estados de `HeuristicAI`

# Parte V. Pruebas y resultados

En esta parte se explican los procesos llevados a cabo para la verificación y validación del producto desarrollado y se presentan los resultados obtenidos usando los agentes de prueba implementados como parte del trabajo.

## 14. Pruebas

### 14.1. Verificación

La verificación del proyecto consiste en comprobar que la implementación responde a su especificación, es decir, que funciona correctamente.

#### Inspección del código

La inspección de código consiste en la revisión del código en busca de errores y aspectos mejorables. A través de la aplicación de esta técnica, se ha podido mejorar la estructura del código de la herramienta de generación de mapas para hacerlo más claro y extensible.

La estrategia para llevar a cabo la inspección ha sido realizar un análisis de la implementación realizada tras haber pasado unas semanas desde la implementación original, de forma que, al no tenerla tan cercana, se consiga una nueva perspectiva más objetiva sobre lo que se había hecho.

#### Pruebas unitarias

Las pruebas unitarias consisten en pruebas a nivel de los funciones o clases del proyecto. En este proyecto, pruebas de este tipo se realizaron durante la fase de desarrollo, eligiendo entre pruebas de tipo «caja negra» y pruebas de tipo «caja blanca» en cada caso según las características específicas de lo que se probaba.

#### Pruebas de integración

Las pruebas de integración buscan verificar la correcta interacción de los componentes de una plataforma. En este caso, las pruebas se fueron realizando incrementalmente según se iban implementando módulos de la plataforma.

#### Pruebas de sistema

Las pruebas de sistema son pruebas que se realizan sobre el sistema completamente integrado.

En primer lugar, dado que el juego no está pensado para que los avatares sean controlados por el usuario, como es el caso en la mayoría de los juegos, se implementó un módulo que se insertó en el proyecto para poder controlar a los avatares y se comprobó que todas las acciones posibles funcionaban correctamente teniendo en cuenta casos típicos y casos límite.

En segundo lugar, se utilizaron los agentes de IA de ejemplo para verificar el correcto funcionamiento del juego, con la idea que estas pruebas complementaban a las anteriores al ser los casos de uso reales y ser además no controladas por el desarrollador directamente, y por tanto más imparciales.

#### Pruebas de la interfaz

En el caso de las interfaces, se realizaron pruebas contemplando todos los posibles casos de uso y flujos del programa.

## 14.2. Validación

La validación de la aplicación consiste en comprobar que satisface el propósito para el que fue planteada. Se comprueba, por tanto, que cumpla los requisitos especificados en la sección 7, tanto los funcionales como los no funcionales.

Para realizar dicha comprobación, se han revisado cada uno de los requisitos funcionales especificados y se ha comprobado que el producto desarrollado implementaba la funcionalidad requerida por los mismos. Por su parte, dada la naturaleza de los requisitos no funcionales de la plataforma, se ha comprobado su satisfacción uno a uno.

Además de la comprobación manual de los requisitos, el desarrollo de agentes de *IA* de ejemplo se ha realizado a modo de prueba de cumplimiento del propósito de la plataforma, pues poder implementar agentes que se puedan inyectar en la plataforma era el objetivo del proyecto. Así pues, se ha realizado una validación experimental de la plataforma.

# 15. Resultados

En esta sección se presentan los resultados de los enfrentamientos realizados entre los agentes de prueba desarrollados. Se han realizado **cinco enfrentamientos** para cada posible emparejamiento los jugadores, en total **quince enfrentamientos**, todos en el mismo mapa, del cual se muestra una imagen en la Figura 37. Los enfrentamientos terminan cuando todos los jugadores de uno de los equipos se quedan sin puntos de vida y duraron entre media hora y una hora cada uno. Véase el apartado 12.8 para una explicación en detalle del sistema de puntuación usado.

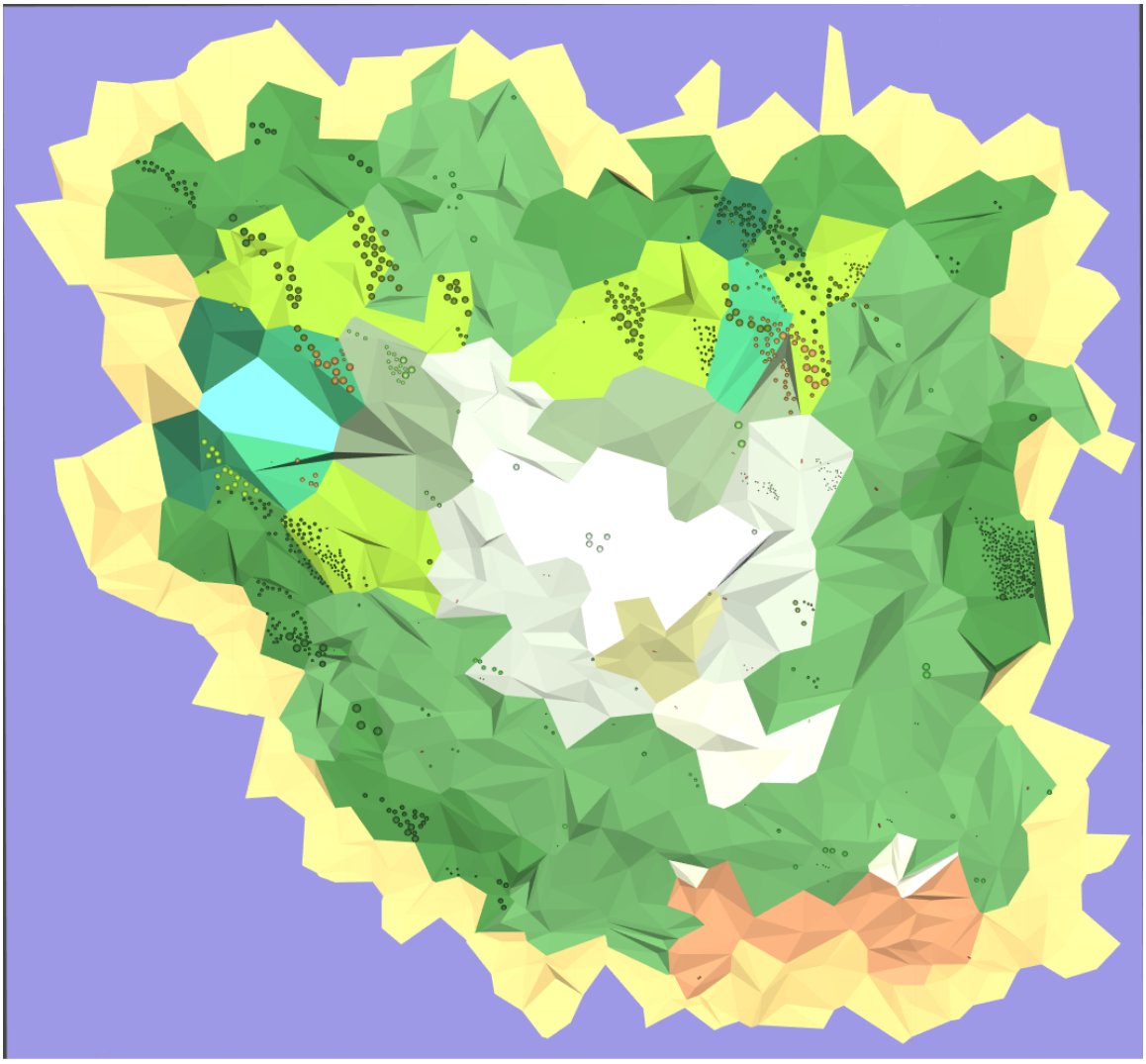


Fig. 37: Mapa usado para las pruebas

En las Tablas del Anexo se muestran los resultados de todos los enfrentamientos y también las medias y desviaciones típicas de cada uno de los emparejamientos.

En la Tabla 7 se resumen los resultados obtenidos a través de la media y la desviación típica de cada uno de los parámetros recogidos por la plataforma durante las partidas para cada uno de los jugadores. Los resultados fueron los esperados: las agentes más elaborados obtuvieron consistentemente mejores resultados.

| | RandomAI | | BasicAI | | HeuristicAI | |
|---|---|---|---|---|---|---|
| MEDIAS DE MEDIDAS | Media | Desviación Típica | Media | Desviación Típica | Media | Desviación Típica |
| Vidas quitadas | 6,9 | 2,657351699 | 18,7 | 1,717556404 | 25 | 0 |
| Cuerpo a Cuerpo | 0,1 | 0,223606798 | 0,2 | 0,273861279 | 3,9 | 1,536364002 |
| Disparando | 6,7 | 2,511610188 | 18,5 | 1,837117307 | 21,1 | 1,536364002 |
| Granadas | 0,1 | 0,223606798 | 0 | 0 | 0 | 0 |
| Con Sigilo | 3,5 | 1,816590212 | 14,2 | 2,830399089 | 15,6 | 2,651414044 |
| A la cabeza | 0 | 0 | 0 | 0 | 0 | 0 |
| Muertes múltiples | 0 | 0 | 0 | 0 | 0 | 0 |
| Fuego amigo | 0 | 0 | 0 | 0 | 0 | 0 |
| Vidas restantes | 0 | 0 | 8,6 | 0,961769203 | 15,8 | 3,4131389 |
| Granadas restantes | 9,7 | 4,667456288 | 22,3 | 1,036822068 | 25 | 0 |
| Puntos por vidas quitadas | 10,4 | 4,306570132 | 33 | 4,537233499 | 40,6 | 2,651414044 |
| Bonus por vida restante | 0 | 0 | 2,7 | 0,273861279 | 5,8 | 0,627414669 |
| Bonus por granadas restantes | 6,6 | 2,119172062 | 10 | 0 | 10 | 0 |
| Bonus por puntería | -2 | 2,567747244 | 5 | 2,439157589 | 5,8 | 2,824880484 |
| Bonus por variedad | 0 | 0 | 0 | 0 | 0 | 0 |
| Puntos totales | 15 | 6,626248368 | 50,6 | 5,920999393 | 62,2 | 4,568296509 |

Tab. 7: Media y desviación típica de los resultados de las partidas de prueba

A continuación se procede a realizar un pequeño análisis de algunos aspectos de los resultados. Lo primero que se puede hacer es estudiar de dónde obtiene cada algoritmo la mayor parte de sus puntos. Los gráficos de la Figura 38 dan una respuesta muy clara: todos los agentes obtienen el grueso de sus puntos directamente de quitar puntos de vida a sus oponentes. La única excepción es en los enfrentamientos entre `RandomAI` y `HeuristicAI`, en los que `RandomAI` aproximadamente la mitad de sus puntos por granadas restantes. Esta excepción puede explicarse observando las bajas puntuaciones que obtiene `RandomAI` contra los otros dos tipos de agente.

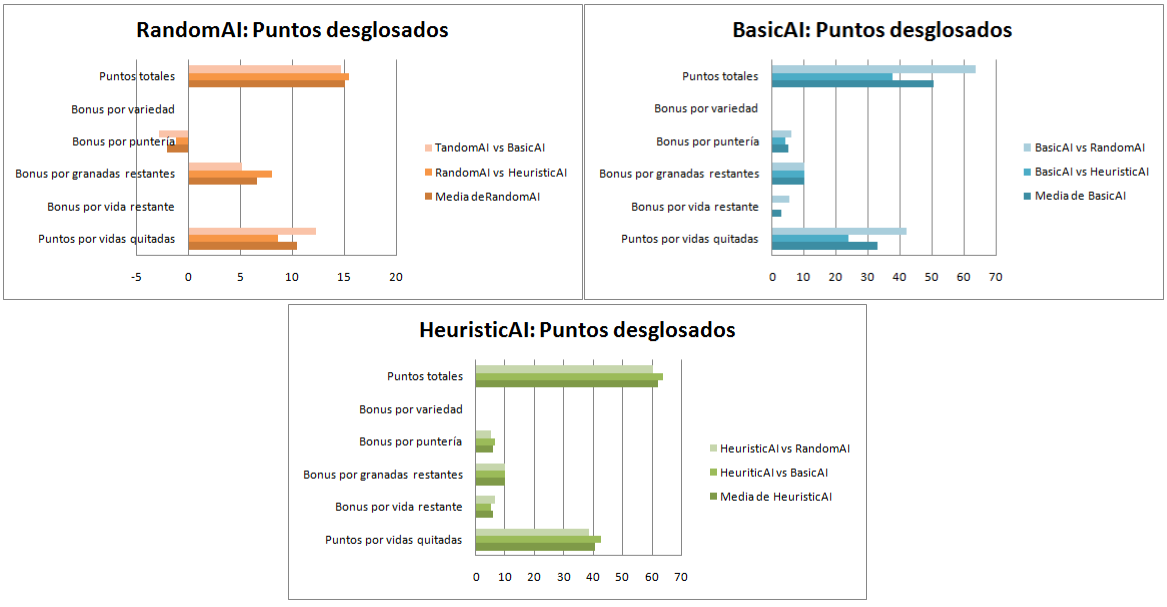



Fig. 38: Desglose de puntos

Dado que existen tres tipos de ataque implementados, es interesante observar que la inmensa mayoría de las muertes se obtuvieron mediante disparos, que sólo `HeuristicAI` consiguió una cantidad sustancial de muertes usando ataques cuerpo a cuerpo y que el uso efectivo de las granadas fue despreciable, como se puede apreciar en la Gráfica 39.

Una consecuencia de que ningún jugador usase con éxito las granadas es que varios de los parámetros usados para obtener las puntuaciones de los equipos no han sido aprovechados. Como la única forma de conseguir *muertes múltiples* es usando granadas para alcanzar a varios adversarios con un mismo ataque y sólo las granadas pueden herir a los aliados, no se produjeron *muertes múltiples* ni casos de *fuego amigo*. Por otra parte, tampoco se obtuvo ningún punto por bonus de *variedad*, pues este parámetro recompensaba precisamente el hecho de usar los tres tipos de ataques disponibles. Por otro lado, el reducido uso de las granadas se tradujo en un bonus estable para `BasicAI` y `HeuristicAI` por *granadas restantes*. Tampoco se aprovechó la posibilidad de obtener bonus por *disparos a la cabeza*.

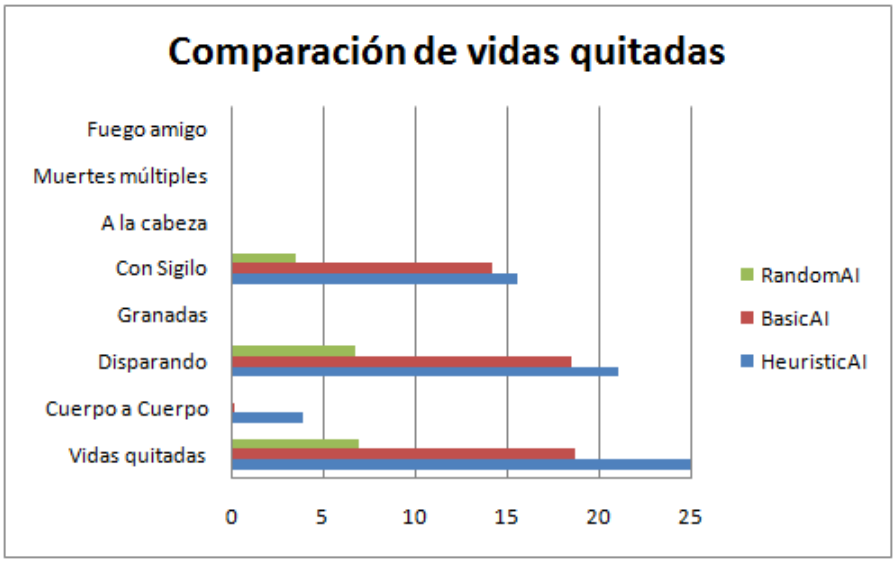



Fig. 39: Gráfico comparativo de las formas de quitar vidas

Este desglose de las vidas quitadas en las formas de quitarlas también permite estudiar las diferencias entre los distintos emparejamientos (Figura 40).

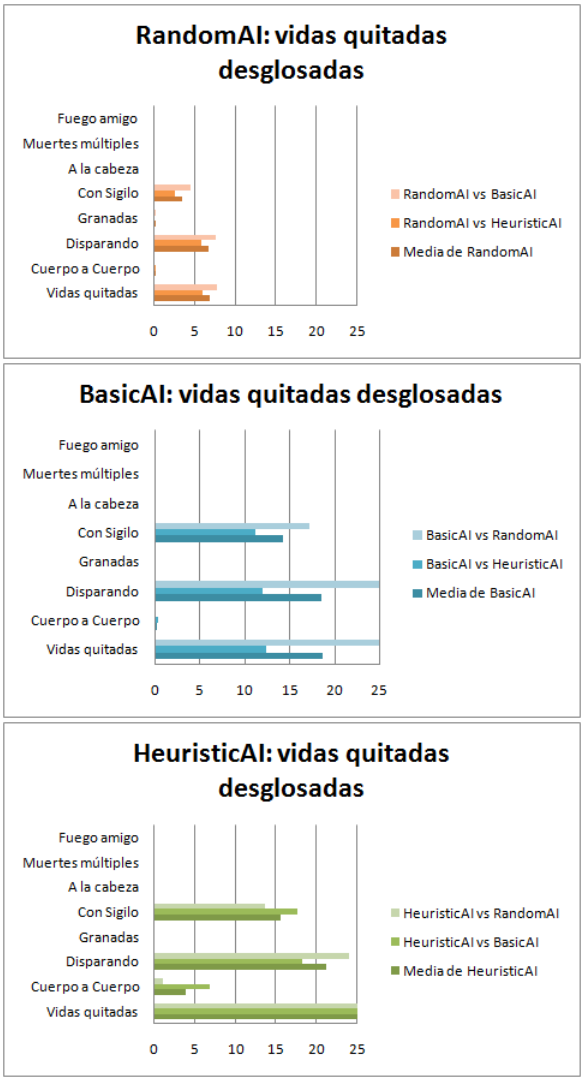



Fig. 40: Desglose de formas de quitar vidas de RandomAI

De nuevo, se observan resultados consistentes: `HeuristicAI` es mejor que `BasicAI`, que es mejor que `RandomAI`. Aunque tanto `HeuristicAI` como `BasicAI` consiguen quitar vidas con ataques cuerpo a cuerpo, es interesante observar que sólo en los enfrentamientos en los que participa `HeuristicAI` se quitan vidas cuerpo a cuerpo y `HeuristicAI` en todos los casos aprovecha mucho más el ataque cuerpo a cuerpo que `BasicAI`. Aún así, la Figura 41 revela que en la mayoría de los encuentros tampoco `HeuristicAI` es capaz de usar el ataque cuerpo a cuerpo contra `RandomAI`. Esto se debe probablemente al comportamiento errático de `RandomAI`, que nunca se persigue a sus oponentes, mientras que `BasicAI` y `HeuristicAI` sí que implementan lógica para acercarse deliberadamente a los oponentes o incluso esperarles en el caso de `HeuristicAI`.

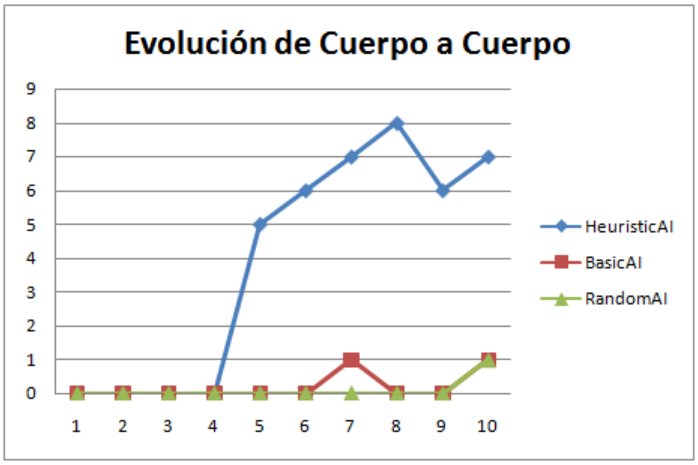



Fig. 41: Gráfica de la evolución de las vidas quitadas con ataques cuerpo a cuerpo

El único bonificador de los disponibles que es aprovechado por los agentes implementados es de de muertes *con sigilo*. Este bonificador se obtiene cuando un jugador alcanza con un ataque a otro antes de que le hayan visto. Aunque ninguno de los agentes implementa estrategias específicamente destinadas a maximizar los puntos obtenidos por este bonificador, puede observarse que `HeuristicAI` consigue obtenerlo en un elevadísimo porcentaje de las ocasiones en ambos emparejamientos y que `BasicAI`, sin embargo consigue resultados bastante peores contra `HeuristicAI`. Esta diferencia podría deberse a la clasificación por prioridades que realiza `HeuristicAI` de los ruidos que hace, que facilita que cuando un adversario le dispare por la espalda, `HeuristicAI` se gire para investigar el origen del disparo, descubriendo así a su asaltante.

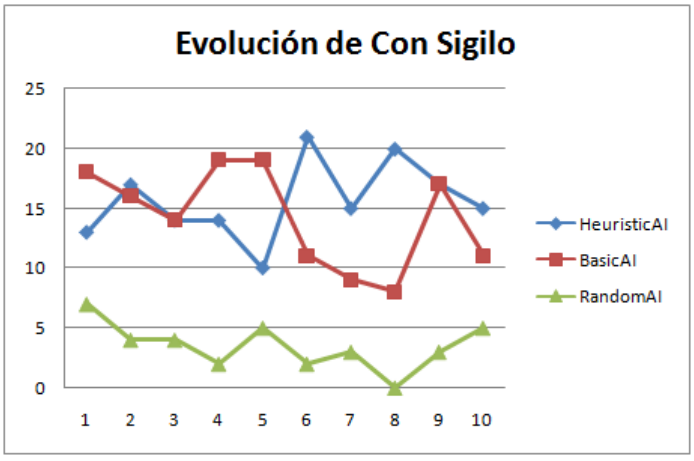



Fig. 42: Gráfico de la evolución del bonificador *Con Sigilo*

El Gráfico 42 muestra con mayor detalle el aprovechamiento del bonificador *Con Sigilo*, haciendo más claro el bajón mucho más pronunciado en el caso de `BasicAI` en este aspecto al enfrentarse a `HeuristicAI`. Por otro lado, también hace ver que ambos obtienen resultados muy parecidos en este

aspecto contra `RandomAI` e incluso algo superiores en el caso de `BasicAI`.

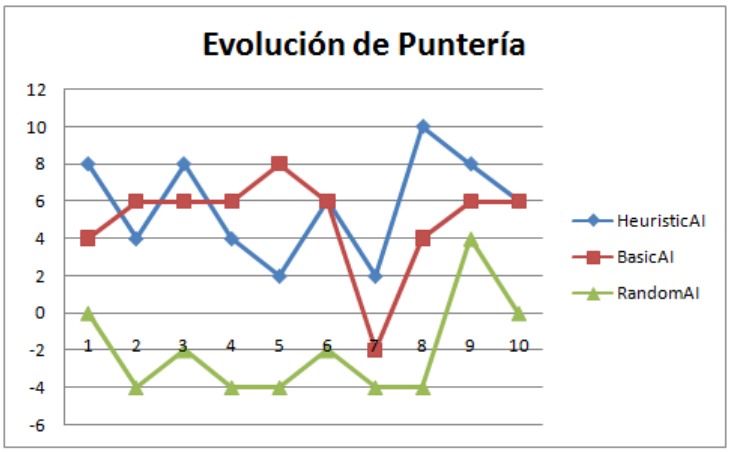



Fig. 43: Gráfico de la evolución del bonificador por *Puntería*

En la Gráfica 43 se ve que, aunque la evolución del bonus de puntería es muy irregular en los tres agentes, `RandomAI` obtiene resultados peores, de acuerdo con el hecho de que nunca invierta tiempo en apuntar ni tenga en cuenta criterio alguno para decidir cuando dejar de apuntar. También de forma coherente con la implementación, `BasicAI` y `HeuristicAI` obtienen resultados parecidos, pues ambos utilizan el algoritmo empleado en el juego para calcular la magnitud de la desviación de los disparos para estimar la probabilidad de acertar y ambos utilizan la misma cota en esta estimación para decidir disparar. No debe sorprender esto, pues la estrategia de `HeuristicAI` no se basa en ser más certero, si no en crear situaciones de superioridad numérica a su favor.

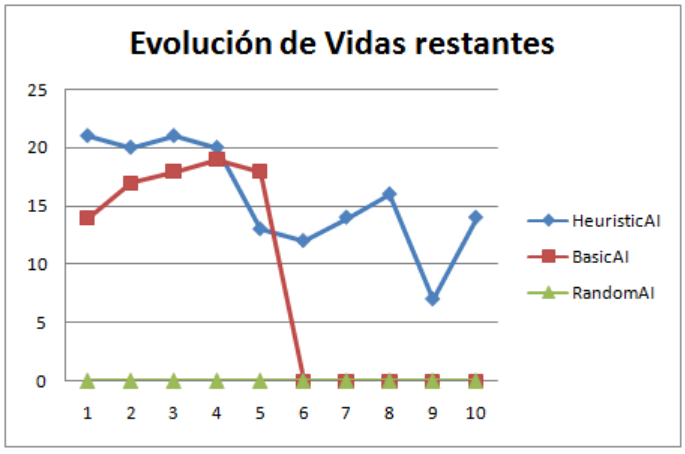



Fig. 44: Gráfico de la evolución de las Vidas restantes

Finalmente, la Gráfica 44, que destaca la evolución de las vidas restantes, sugiere una alternativa en el formato de los enfrentamientos. Aunque en este caso, ninguno de los agentes participantes implementaban tácticas de supervivencia, ésta podría haber sido una estrategia interesante usada

por otros agentes, y en ese caso, el hecho de extender las partidas hasta que todos los jugadores de uno de los equipos fueran eliminados hubiese supuesto una desventaja notoria para ellos. Por tanto, sería interesante cambiar el criterio de finalización de los enfrentamientos, quizás dando varias condiciones distintas que marcasen el fin de la partida, o simplemente estableciendo un límite de tiempo, pues de esta forma, se bonifica doblemente al equipo que elimina a todos los jugadores oponentes: primero por ser el que más vidas ha quitado, y después porque siempre tendrán un bonificador por vidas restantes mayor.

# Parte VI. Conclusiones

Como se estableció en los objetivos del trabajo, se ha creado una plataforma que ofrece unas características diferentes de las ofrecidas por las otras plataformas existentes en el ámbito. Aunque se han cumplido los objetivos del trabajo y se ha creado un producto enteramente funcional, ya se adelantó en el apartado 5 que el alcance del proyecto es más amplio que el de este trabajo. El trabajo desarrollado, especialmente la fase de desarrollo de los agentes de prueba, ha dado una nueva perspectiva acerca de ciertos aspectos de la plataforma, y por tanto ha sugerido numerosas ampliaciones posibles. No se considera esto, sin embargo, un indicio de fracaso, si no una situación deseable. Desde el principio se entendió que este proyecto debía ser algo vivo, que no podría realizarse más que en incrementos sucesivos basados en la experiencia de uso de la plataforma después de cada nuevo incremento. Por este motivo, en esta sección se proponen posibles lineas de desarrollo futuras. Dada la cantidad de aspectos de la plataforma en los que se han observado opciones de mejora, se separan las ampliaciones según su prioridad e interés.

## 16. Ampliaciones Prioritarias

### 16.1. Juego

- Aunque se realizó una inspección de código de la herramienta de generación de mapas, esto nunca llegó a hacerse con el componente de juego en sí mismo. Dada la importancia de la extensibilidad de la implementación de la plataforma, esta sería una de los primeras vías de mejora, pues ayudaría a mejorar la estructura del código, mejorando, por ejemplo aspectos como la encapsulación, y con ella el control del acceso de los agentes a partes del código a las que no deben acceder.
- La implementación actual de la interfaz para los agentes y el método para darles tiempo de ejecución permite que implementen algoritmos no atómicos, es decir, que tomen decisiones - posiblemente más elaboradas - utilizando varios «turnos» para ello (ver distinción entre juego dinámico y estático en el apartado 2 o en [12]). Este aspecto podría mejorarse si se aprovechasen las características de las corutinas para permitir la interrupción de la ejecución de algoritmos no atómicos al final de los turnos y la continuación de su ejecución en el siguiente turno. Esta gestión automática de agentes que tomen decisiones no atómicamente facilitaría el aprovechamiento de esta característica para los desarrolladores. La implementación actual del juego está ya orientada a facilitar esta ampliación.

## 17. Ampliaciones interesantes

### 17.1. Generador de mapas

- Para gestionar los casos degenerados en los que los ríos no desembocan al mar, se podrían crear lagos en los extremos de dichos ríos, usando por ejemplo una versión adaptada del método de la isla radial usado en el algoritmo implementado.

- La distribución de los troncos caídos en el terreno se podría mejorar utilizando la distribución de los árboles para condicionar su posicionamiento en vez de posicionarlos de forma esencialmente aleatoria en el mapa. Así mismo, podrían enriquecerse las formaciones de vegetación sin añadir nuevos objetos de terreno simplemente estableciendo relaciones condicionantes entre la presencia de árboles y arbustos.

- Aunque los objetos del terreno ya utilizados proveen obstrucción a la visión y a los disparos, la inclusión de muros y pequeños edificios en ruinas definiría puntos estratégicos más claros, lo que reforzaría la importancia de la mecánica de análisis del terreno.

## 17.2. Juego

- Aunque el juego incluye un sistema de puntuación de los equipos que se usa para determinar qué equipo gana cada partida, actualmente no existen condiciones que definan el fin de las partidas. En las pruebas realizadas con los agentes desarrollados las partidas acababan cuando todos los jugadores de uno de los equipos participantes perdían todos los puntos de vida. Una alternativa a esta opción que se discutió en el apartado **??** es fijar una duración para las partidas y tomar las puntuaciones de los equipos en ese momento como los resultados de la partida. Sin embargo, el juego podría ofrecer explícitamente al usuario la selección de estas y otras opciones de finalización de partida, incluyendo, por ejemplo el cumplimiento de ciertos objetivos, lo que añadiría una capa estratégica adicional al juego.

- En la versión actual, los agentes no tienen acceso a las puntuaciones de los equipos ni de los jugadores en ningún momento de la partida. Sin embargo, el acceso a esta información podría ser útil para la toma de decisiones por parte de los agentes.

- Actualmente, la plataforma posibilita la implementación de agentes individuales e independientes para controlar a los jugadores, así como el uso de jerarquías para definir *roles* e interacciones específicas entre jugadores con papeles concretos. Sin embargo, un posible modo alternativo de partida sería que un mismo agente inteligente recibiese toda la información de sus agentes y los controlase a todos, a modo de general cuasi-omnisciente.

- El número de equipos y el número de agentes de cada equipo podría hacerse configurable para cada partida, posibilitando mecánicas de alianzas y traiciones entre los equipos.

- La incorporación de un sistema de niebla de batalla (*fog of war*) [42] y la aparición de la necesidad para los agentes de definir mecanismos para elegir una ruta entre dos puntos entre los que no se conoce con certeza la ruta óptima aumentaría el peso de la faceta de exploración del juego, así como la del análisis del terreno.

- Basándose en los resultados obtenidos en las primeras pruebas realizadas con los agentes de prueba desarrollados, podrían realizarse cambios en el sistema de puntuación de las partidas, así como cambios en algunos de las características ya implementadas en el juego, por ejemplo la forma de uso de las granadas.

- La inclusión de un sistema de atributos para los jugadores que permitiese formar equipos con jugadores con diferentes características (*e.g.* puntería, resistencia, velocidad, sigilo, visión, oído...), entre otras cosas, aumentaría el interés del uso de las jerarquías y *roles* en las estrategias de los equipos.

- La inclusión de un sistema de energía que asigne un coste energético a las acciones de los jugadores, añadiendo un nuevo método para regular el uso de ciertas acciones y un nuevo factor a considerar para los agentes.

- Aunque inicialmente se optó por no usar un sistema de disparo clásico - y determinista - puramente basado en la dirección en la que apunten los jugadores y el sistema de colisiones para determinar la trayectoria de los disparos y si estos impactaban o no a otros jugadores, se ha observado que este sistema podría simplificar la interfaz para los agentes y ofrecer nuevas posibilidades sin perder las ya existentes. Entre otras cosas, el sistema clásico añadiría la posibilidad de que se produjese fuego amigo accidental, al no tener que fijar a un objetivo para dispararle y permitiría el uso de heurísticas para realizar disparos con posibilidad de impactar a enemigos que no estuviesen en el rango de visión del jugador en el momento del disparo. Por otro lado, en combinación con la implementación de armas con diferentes características que favorezcan diferentes tácticas de uso y exijan diferentes métodos para apuntar [43] (disparos con distinta velocidad, armas con distintos alcances, armas con distintas precisiones, armas con distinta cantidad de munición, armas con distinta velocidad de disparo (*fire rate*)...) permitirían conservar el factor de indeterminismo en los disparos y favorecerían la definición de *roles* y por tanto la cooperación entre los integrantes de los equipos.

- La inclusión de métodos más sólidos para la determinación de la visibilidad en el motor del juego y de métodos más fiables para la predicción de visibilidad en el entorno en la *API* ofrecida a los agentes facilitaría la implementación de algoritmos de análisis de terreno a los usuarios al mejorar la detección de puntos estratégicos en el mapa, como por ejemplo zonas de cobertura.

# 18. Ampliaciones secundarias

## 18.1. Generador de mapas

- El generador la superficie sólo ofrece un método para definir la forma de la isla. El hecho de que todos los mapas sean islas es una decisión de diseño que se sigue considerando acertada, pero la inclusión de nuevos algoritmos para definir la forma de la isla podría aportar nuevos aspectos estratégicamente relevantes a los mapas. En concreto, se ha estudiado la posibilidad de crear islas utilizando *ruido Perlin* [37, 38, 24], ampliamente usado en el contexto de la generación procedural de terreno o permitir la posibilidad de generar la forma de una isla a partir de una imagen aportada por el usuario que se usaría como máscara de bits.

- Utilizar el algoritmo de *relajación de Lloyd* [39] para aumentar la regularidad del grafo creado con el *algoritmo de Voronoi*.

- Añadir más métodos para definir la elevación. Concretamente, el uso de *2D pink noise* [29], *midpoint displacement*, fractales [41], *diamond-square* [40] o *ruido Perlin* es una práctica extendida en el ámbito que produce buenos resultados.

- El algoritmo de generación de objetos del terreno se ejecuta de forma síncrona en su implementación actual y es sin embargo un proceso que tarda generalmente varios segundos. Como *Unity3D* se ejecuta en un sólo hilo, esto se traduce en una congelación de la aplicación durante la fase de generación de objetos que puede resultar desconcertante y molesta para el usuario. Haciendo uso de la funcionalidad de las corutinas [22] que ofrece *Unity3D*, el algoritmo de generación de objetos podría ejecutarse de forma asíncrona.

## 18.2. Juego

- La sustitución del sistema de registro de las acciones de los jugadores en las partidas implementado crea *logs* en formato *XML* que ocupan una gran cantidad de espacio debería sustituirse por otro que cree *logs* más comprimidos.

- Incluir un sistema de audio que de *feedback* auditivo a los usuarios de las acciones de los agentes en el juego (disparos, movimientos, comunicaciones, etc...) , así como completar el apartado gráfico con mejores modelos y animaciones e implementar la representación gráfica de la pintura en el juego.

## Referencias

[1] Turing, A. (1950) en Thompson, T. (2014). Why Study AI in Games? `http://t2thompson.com/2014/01/26/why-ai-in-games/`

[2] Koza, J. (1992) y Rosca, J. (1996) en Thompson, T. (2014). What's the Deal with Pac-Man? `http://t2thompson.com/2014/02/10/ai-and-pacman/`

[3] Lucas, S. (2007). Conference on Evolutionary Computation (CEC) `http://www.cec2015.org/`

[4] Lucas, S. (2009;2011). IEEE Conference on Computational Intelligence and Games (IEEE CIG). Ms. Pac-Man AI Competition (2009) y Ms. Pac-Man vs Ghosts Competition (2011). `http://cig2015.nctu.edu.tw/`

[5] Togelius, J. , Karakovskiy, S., Shaker, N. (2009;2010). Mario AI Competition. `http://www.marioai.org/home`

[6] Polygon, 2015. Telepath Tactics mixes Fire Emblem with... 19th-century Prussian military theory? [Video]. Disponible en `http://www.polygon.com/2015/4/29/8514995/telepath-tactics-gameplay-overview-lets-play`

[7] Wikipedia. Kriegsspiel (wargame). `http://en.wikipedia.org/wiki/Kriegsspiel_%28wargame%29`

[8] Kim, Kyung-Joong, Cho, Ho-Chul, Oh y In-Suk (2010-2014). IEEE Computational Intelligence and Games (IEEE CIG). Starcraft AI Competition. http://cilab.sejong.ac.kr/sc_competition/

[9] Shaker, N. (2013). Platformer AI Competition. http://www.platformersai.com/

[10] Hingston, P., Polceanu, M. (2008-2011). 2k BotPrize. Can computers play like people? http://botprize.org/index.html

[11] Desmoulins, F., Antoniazzi, N., Barral, A. et al. (2012-2015). CodinGame. https://www.codingame.com/home

[12] Michard, L. (2015). Practicing Artificial Intelligence Programming.https://www.codingame.com/blog/en/2015/04/practice-artificial-intelligence-programming-codingame.html

[13] van der Sterren, W. (2001). Terrain Reasoning for 3d Action Games. http://www.gamasutra.com/view/feature/131447/terrain_reasoning_for_3d_action_.php

[14] Unity3D. https://unity3d.com/es

[15] Blender. https://www.blender.org/

[16] Procedural generation. http://en.wikipedia.org/wiki/Procedural_generation

[17] Unity Documentation. Manual: Meshes. http://docs.unity3d.com/Manual/class-Mesh.html

[18] Unity Documentation. Manual: Creating Scenes. http://docs.unity3d.com/460/Documentation/Manual/CreatingScenes.html

[19] Unity Documentation. Manual: Editor. http://docs.unity3d.com/Manual/Editor.html

[20] Unity Documentation. Manual: Navigation. http://docs.unity3d.com/Manual/Navigation.html

[21] Unity Documentation. Manual: Bounds. http://docs.unity3d.com/ScriptReference/Bounds.html

[22] Unity Documentation. Manual: Coroutines http://docs.unity3d.com/Manual/Coroutines.html

[23] Wikipedia: Frame http://es.wikipedia.org/wiki/Frame#En_inform.C3.A1tica

[24] Patel, A. (2010). Polygonal Map Generation for Games. http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/

[25] «Jayelinda». (2013). Modelling by numbers. An introduction to procedural geometry. http://jayelinda.com/modelling-by-numbers-part-1a/

[26] Sucasas, A. L. (2015). El País. Las ventas de videojuegos doblan a la taquilla de cine en España. http://cultura.elpais.com/cultura/2015/03/25/actualidad/1427309707_733302.html

[27] Wikipedia: Voronoi Diagram. http://en.wikipedia.org/wiki/Voronoi_diagram

[28] «jceipek» (2014). Unity-delaunay. https://github.com/jceipek/Unity-delaunay

[29] Padel, A. (2013). Noise functions and map generation. http://www.redblobgames.com/articles/noise/introduction.html

[30] Whittaker, R. H. (1975). Whittaker Biome Diagram. http://www.marietta.edu/~biol/biomes/biome_main.htm

[31] «Pomax» (Unknown). A Primer on Bézier Curves. pomax.github.io/bezierinfo

[32] Unity Documentation. Manual: Colliders. http://docs.unity3d.com/Manual/CollidersOverview.html

[33] Unity Tutorials. Manual: Raycasting. [Video]. Disponible en https://unity3d.com/es/learn/tutorials/modules/beginner/physics/raycasting

[34] Unity Documentation. Manual: Raycast. http://docs.unity3d.com/ScriptReference/Physics.Raycast.html

[35] Unity Documentation. Manual: UI. http://docs.unity3d.com/Manual/UISystem.html

[36] Simpson, C. (2014). Behavior trees for AI: How they work. http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

[37] Wikipedia: Perlin Noise. http://en.wikipedia.org/wiki/Perlin_noise

[38] Tulleken, H. (2009). How to use Perlin Noise in Your Games. http://devmag.org.za/2009/04/25/perlin-noise/

[39] Wikipedia: Lloyd's algorithm. http://en.wikipedia.org/wiki/Lloyd's_algorithm

[40] Wikipedia: Diamond.square algorithm. http://en.wikipedia.org/wiki/Diamond-square_algorithm

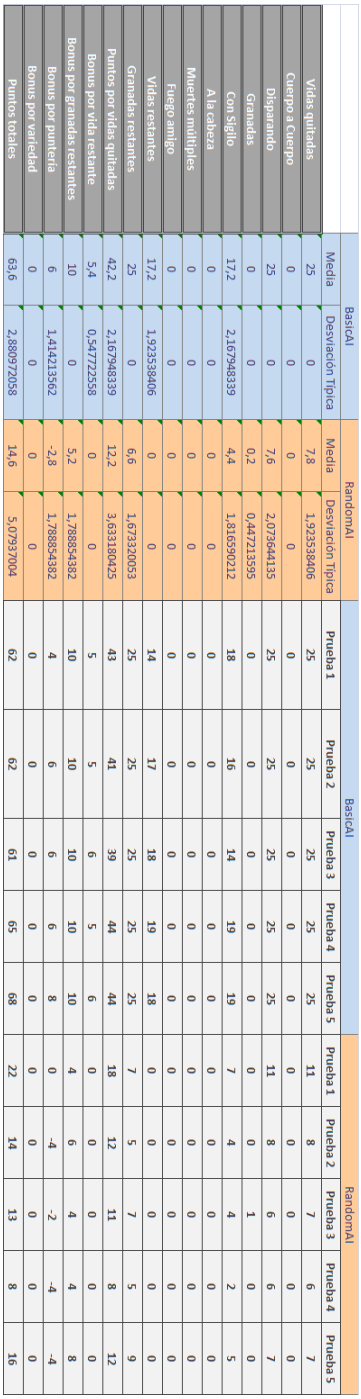
[41] Martz, P. (1996). Generating Random Fractal Terrain. http://www.gameprogrammer.com/fractal.html

[42] «daveyd». (2014). Giant Bomb: Fog of War concept. http://www.giantbomb.com/fog-of-war/3015-14/

[43] Sanchez-Crespo, D. (2003). Game Programming: Action-Oriented AI. [Página 8] http://www.peachpit.com/articles/article.aspx?p=102090&seqNum=8

# Anexos

## Tablas completas de los enfrentamientos



| | BasicAI | | RandomAI | | BasicAI | | | | | RandomAI | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Media | Desviación Típica | Media | Desviación Típica | Prueba 1 | Prueba 2 | Prueba 3 | Prueba 4 | Prueba 5 | Prueba 1 | Prueba 2 | Prueba 3 | Prueba 4 | Prueba 5 |
| Vidas quitadas | 25 | 0 | 7,8 | 1,923538406 | 25 | 25 | 25 | 25 | 25 | 11 | 8 | 7 | 6 | 7 |
| Cuerpo a Cuerpo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Disparando | 25 | 0 | 7,6 | 2,073644135 | 25 | 25 | 25 | 25 | 25 | 11 | 8 | 6 | 6 | 7 |
| Granadas | 0 | 0 | 0,2 | 0,447213595 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Con Sigilo | 17,2 | 2,167948339 | 4,4 | 1,816590212 | 18 | 16 | 14 | 19 | 19 | 7 | 4 | 4 | 2 | 5 |
| A la cabeza | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Muertes múltiples | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fuego amigo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Vidas restantes | 17,2 | 1,923538406 | 0 | 0 | 14 | 17 | 18 | 19 | 18 | 0 | 0 | 0 | 0 | 0 |
| Granadas restantes | 25 | 0 | 6,6 | 1,673320053 | 25 | 25 | 25 | 25 | 25 | 7 | 5 | 7 | 5 | 9 |
| Puntos por vidas quitadas | 42,2 | 2,167948339 | 12,2 | 3,633180425 | 43 | 41 | 39 | 44 | 44 | 18 | 12 | 11 | 8 | 12 |
| Bonus por vida restante | 5,4 | 0,547722558 | 0 | 0 | 5 | 5 | 6 | 5 | 6 | 0 | 0 | 0 | 0 | 0 |
| Bonus por granadas restantes | 10 | 0 | 5,2 | 1,788854382 | 10 | 10 | 10 | 10 | 10 | 4 | 6 | 4 | 4 | 8 |
| Bonus por puntería | 6 | 1,414213562 | -2,8 | 1,788854382 | 4 | 6 | 6 | 6 | 8 | 0 | -4 | -2 | -4 | -4 |
| Bonus por variedad | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Puntos totales | 63,6 | 2,880972058 | 14,6 | 5,079370044 | 62 | 62 | 61 | 65 | 68 | 22 | 14 | 13 | 8 | 16 |

Tab. 8: BasicAI contra RandomAI

| | HeuristicAI | | RandomAI | | HeuristicAI | | | | | RandomAI | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Media | Desviación Típica | Media | Desviación Típica | Prueba 1 | Prueba 2 | Prueba 3 | Prueba 4 | Prueba 5 | Prueba 1 | Prueba 2 | Prueba 3 | Prueba 4 | Prueba 5 |
| Vidas quitadas | 25 | 0 | 6 | 3,391164992 | 25 | 25 | 25 | 25 | 25 | 4 | 5 | 4 | 5 | 12 |
| Cuerpo a Cuerpo | 1 | 2,236067977 | 0,2 | 0,447213595 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 1 |
| Disparando | 24 | 2,236067977 | 5,8 | 2,949576241 | 25 | 25 | 25 | 25 | 20 | 4 | 5 | 4 | 5 | 11 |
| Granadas | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Con Sigilo | 13,6 | 2,50998008 | 2,6 | 1,816590212 | 13 | 17 | 14 | 14 | 10 | 2 | 3 | 0 | 3 | 5 |
| A la cabeza | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Muertes múltiples | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fuego amigo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Vidas restantes | 19 | 3,391164992 | 0 | 0 | 21 | 20 | 21 | 20 | 13 | 0 | 0 | 0 | 0 | 0 |
| Granadas restantes | 25 | 0 | 12,8 | 7,661592524 | 25 | 25 | 25 | 25 | 25 | 5 | 14 | 12 | 8 | 25 |
| Puntos por vidas quitadas | 38,6 | 2,50998008 | 8,6 | 4,979959839 | 38 | 42 | 39 | 39 | 35 | 6 | 8 | 4 | 8 | 17 |
| Bonus por vida restante | 6,6 | 0,547722558 | 0 | 0 | 7 | 6 | 7 | 7 | 6 | 0 | 0 | 0 | 0 | 0 |
| Bonus por granadas restantes | 10 | 0 | 8 | 2,449489743 | 10 | 10 | 10 | 10 | 10 | 4 | 10 | 8 | 8 | 10 |
| Bonus por puntería | 5,2 | 2,683281573 | -1,2 | 3,346640106 | 8 | 4 | 8 | 4 | 2 | -2 | -4 | -4 | 4 | 0 |
| Bonus por variedad | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Puntos totales | 60,4 | 4,393176527 | 15,4 | 8,173126697 | 63 | 62 | 64 | 60 | 53 | 8 | 14 | 8 | 20 | 27 |

Tab. 9: HeuristicAI contra RandomAI

| | HeuristicAI | | BasicAI | | HeuristicAI | | | | | BasicAI | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Media | Desviación Típica | Media | Desviación Típica | Prueba 1 | Prueba 2 | Prueba 3 | Prueba 4 | Prueba 5 | Prueba 1 | Prueba 2 | Prueba 3 | Prueba 4 | Prueba 5 |
| Vidas quitadas | 25 | 0 | 12,4 | 3,435112807 | 25 | 25 | 25 | 25 | 25 | 13 | 11 | 9 | 18 | 11 |
| Cuerpo a Cuerpo | 6,8 | 0,836660027 | 0,4 | 0,547722558 | 6 | 7 | 8 | 6 | 7 | 0 | 1 | 0 | 0 | 1 |
| Disparando | 18,2 | 0,836660027 | 12 | 3,674234614 | 19 | 18 | 17 | 19 | 18 | 13 | 10 | 9 | 18 | 10 |
| Granadas | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Con Sigilo | 17,6 | 2,792848009 | 11,2 | 3,492849839 | 21 | 15 | 20 | 17 | 15 | 11 | 9 | 8 | 17 | 11 |
| A la cabeza | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Muertes múltiples | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fuego amigo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Vidas restantes | 12,6 | 3,435112807 | 0 | 0 | 12 | 14 | 16 | 7 | 14 | 0 | 0 | 0 | 0 | 0 |
| Granadas restantes | 25 | 0 | 19,6 | 2,073644135 | 25 | 25 | 25 | 25 | 25 | 21 | 17 | 22 | 20 | 18 |
| Puntos por vidas quitadas | 42,6 | 2,792848009 | 23,8 | 6,90851866 | 46 | 40 | 45 | 42 | 40 | 25 | 20 | 17 | 35 | 22 |
| Bonus por vida restante | 5 | 0,707106781 | 0 | 0 | 5 | 6 | 5 | 4 | 5 | 0 | 0 | 0 | 0 | 0 |
| Bonus por granadas restantes | 10 | 0 | 10 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Bonus por puntería | 6,4 | 2,966479395 | 4 | 3,464101615 | 6 | 2 | 10 | 8 | 6 | 6 | -2 | 4 | 6 | 6 |
| Bonus por variedad | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Puntos totales | 64 | 4,743416491 | 37,6 | 8,961026727 | 67 | 58 | 70 | 64 | 61 | 40 | 28 | 31 | 51 | 38 |

Tab. 10: HeuristicAI contra BasicAI